UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

FRANCISCO GERDAU DE BORJA

Framework Flexível para Produtos Financeiros de Alto Desempenho

Trabalho de Graduação.

Prof. Dr. Cláudio Fernando Resin Geyer
Orientador

M. Sc. Christian de Schryver
Me. Julio Anjos
Co-orientadores

Porto Alegre, janeiro de 2013

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

de Borja, Francisco Gerdau

Framework Flexível para Produtos Financeiros de Alto Desempenho / Francisco Gerdau de Borja. – Porto Alegre: Graduação em Ciência da Computação da UFRGS, 2013.

64 f.: il.

Trabalho de Conclusão (bacharelado) – Universidade Federal do Rio Grande do Sul. BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO, Porto Alegre, BR–RS, 2013. Orientador: Cláudio Fernando Resin Geyer; Co-orientadores: Christian de Schryver
Me. Julio Anjos.

1. Framework. 2. Financeiro. 3. Computação distribuída. I. Geyer, Cláudio Fernando Resin. II. Me. Julio Anjos, Christian de Schryver
. III. Título.

“Se eu pudesse deixar algum sentimento a você, deixaria aceso o sentimento de amar a vida dos seres humanos. Deixaria a consciência de aprender tudo o que foi ensinado pelo tempo a fora. Lembraria os erros que foram cometidos para que não mais se repetissem. Deixaria para você, se pudesse, o respeito àquilo que é indispensável: além do pão, o trabalho. Além do trabalho, a ação. E quando tudo, por acaso, lhe faltasse, um segredo: o de buscar no interior de si mesmo a resposta e a força para encontrar a saída.”

— MAHATMA GANDHI

AGRADECIMENTOS

À minha família, aos meus amigos, aos meus amores e aos meus mestres.

Família cerne de minha educação, recebi desta liberdade para escolher minha direção mas sempre esteve presente para soprar ventos bem-vindos, aprumando o rumo longe de caminhos tortuosos.

Amigos na alegria e na tristeza, na saúde e na doença, na riqueza e na pobreza, por todos os dias de minha vida, até que a morte nos separe.

Amores, sim amei e amo! “porque a vida só se dá pra quem se deu”... especialmente à Júlia que me acompanhou no fim deste trabalho.

Mestres da excelentíssima UFRGS, professor Geyer por ter me acolhido em seu grupo de pesquisa, à TU Kaiserslautern e grupo de microeletrônica, professor When e orientador Christian de Schryver.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

EU	União Europeia
GPGPU	General Purpose Graphic Processor Unit
FPGA	Field-Programmable Gate Array
SOA	Services-Oriented Architecture
RDMA	Remote Direct Memory Access
FIX	Financial Information Exchange
AMQP	Advanced Message Queueing
EDA	Event-Driven Architecture
MOM	Message-Oriented Middleware
CBD	Component-Based Development
OOD	Object-Oriented Development
RUP	Rational Unified Process
ACK	Acknowledgement
TTL	Time To Live
FIFO	First In, First Out
APIs	Application Programming Interface
PID	Process Indentification Number

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A crise financeira que abalou as economias em 2008, demonstrou a fragilidade do mercado em relação as agências de rating financeiro. Duas lições foram aprendidas - a necessidade de regulamentar a quantidade de risco máximo sobre investimentos – e a necessidade de tornar-se independente na avaliação de riscos. A avaliação de riscos sobre investimentos é uma tarefa complexa pois é derivada da sinergia de diversos fatores. Esta complexidade somada ao requisito de baixa latência do mercado, torna os sistemas de avaliação caríssimos, logo apenas disponíveis a grandes empresas.

A proposta deste trabalho visa compensar este desnível tecnológico, é apresentado um framework acessível para empresas, investidores e pesquisadores criarem avaliações financeira de alto desempenho. Para atender a baixa latência exigida pelo mercado, o framework utiliza troca de mensagens assíncronas em uma arquitetura distribuída orientada a serviços-eventos. A flexibilidade e acessibilidade são garantidas com padrões abertos de interoperabilidade e integração.

Paralelamente, é feito um estudo da avaliação de um instrumento financeiro complexo, as opcões, utilizando o modelo Black-Scholes e o método Monte Carlo. Por fim é avaliado o *speedup* da implementação da avaliação de opções sobre o framework. Complementarmente é apresentado o ProcessStarter, um sistema para iniciar, gerenciar e monitorar processos remotos.

Palavras-chave: Framework, financeiro, computação distribuída.

1 INTRODUÇÃO

1.1 Ambientação

Nos anos seguintes a crise financeira de 2008, que teve seu epicentro nos Estados Unidos da América, o mercado foi analisado minuciosamente no intuito de achar "culpados". Esta investigação, desenvolvida sobre os olhares da comunidade internacional, culminou no *'Anatomy of a Financial Collapse'* (Senate 2011), este documento aponta que: A crise não foi um desastre natural, mas o resultado de alto risco sobre produtos financeiros complexos. Mas como hipotecas, um dos mais antigos instrumentos financeiros (Homer e Sylla 1996) (Kaelber 2004) (Smith 1927) se tornaram produtos financeiros complexos de alto risco? Com derivativos; derivativos sobreavaliados acabaram se tornando populares por seu alto retorno monetário mas escondiam um alto risco. Segundo o senado americano o risco desses fundos foi causado por analistas financeiros apressados, sobrecarregados e desmoralizados. Foram incumbidos de avaliar números crescentes de instrumentos financeiros cada vez mais complexos em alta velocidade, usando modelos de avaliação antiquados e critérios de avaliação incertos. Estes problemas estão relacionados com o fato destes avaliadores serem majoritariamente pertencentes as agências de *rating* que dominam o mercado de crédito americano, conhecidas como *Big Three* [1], o relatório do senado aponta como um dos fatores o comodismo destas empresas.

Após 2008, bancos pertencentes ao G-10 aderiram a um *framework* comum para avaliação e gerência de riscos, o Basel II, e consequentemente, a crise financeira na EU, a adoção do Basel III (Ennis e Price 2011). (Ying 2011) e (Caruana 2010) citam a importância destes acordos e leis para os investidores e países em desenvolvimento, como o Brasil, entretanto Ying reforça a ideia de que o desenvolvimento de avaliadores (*rating agencies*) domésticos é uma questão de segurança nacional, e estes irão se desenvolver assim que o seu sistema financeiro se aprofundar em pesquisas acadêmicas.

Com a popularização dos computadores e internet os investidores alcançaram o fácil acesso as informações e um crescente poder computacional, portanto ganharam uma poderosa ferramenta de análise. (Ellis 1997) mostra as mudanças causadas nos perfis dos investidores, com estes recursos eles tendem a criar suas próprias classificações usando as informações das agências avaliadoras como complemento em seus cálculos e decisões.

Apesar do ganho adquirido pelos computadores pessoais, a alta demanda computacional de ferramentas financeiras, que têm complexidade computacional variando entre problemas de tempo polinomial e NP-hard, tem sua eficiência limitada a uma escalabilidade pequena sobre estes processadores multinúcleos. Para tais tarefas a indústria

[1] Moody's Investors Service, Inc. (Moody's); Standard & Poor's Financial Services LLC (S&P); and Fitch Ratings Ltd. (Fitch). By some accounts, these firms issue about 98% of the total credit ratings and collect 90% of total credit rating revenue in the United States. (Senate 2011)

financeira usa software proprietário, bem como comercial voltada ao uso de GPGPU, e FPGA (Irturk et al. 2008). Mas a fim de utilizar eficientemente as centenas de *gigaflops* oferecidas por esses sistemas, o programador da aplicação requer maior esforço para a paralelização e um maior investimento monetário.

Estes fatores criam uma grande lacuna tecnológica, entre as grandes empresas financeiras, bancos e *rating agencies*, em oposição a outros participantes do mercado. A proposta deste trabalho visa compensar este desnível tecnológico, é apresentado um *framework* acessível para pequenas empresas, investidores e pesquisadores criar componentes de avaliação financeira de alto desempenho usando tecnologias abertas de integração. A seguir são apresentados os desafios e estado da arte em sistemas financeiros. Continuando na Seção 1.3, introduzimos a aplicação que será testada sobre o *framework*, a avaliação de um comum instrumento financeiro complexo, as opções.

1.2 Arquiteturas financeiras : desafios e tendências

Arquiteturas de próxima geração têm de responder a exigências crescentes para velocidade, volume e eficiência. Por exemplo, no mercado de moedas, 75% do volume são negociações *intra-daily* e nele populariza-se os sub-second trades, abertura e fechamento de negócios perto do limite da percepção humana (Liukkonen 2009) (Saariluoma 1995). O investidor tornou-se um "engenheiro financeiro", pois aplica seus conhecimentos para ajustar seus modelos comerciais *on the fly*. Empresas de desenvolvimento que visam novos instrumentos financeiros, como derivativos, precisam implementar as novas aplicações rapidamente e de forma escalável (Cisco 2008).

A Figura 1.1 mostra uma pesquisa feita pela IBM que peguntou a executivos financeiros: “quais são os principais impedimentos para sua empresa executar suas estratégias de negócios nos próximos 10 anos”. Nota-se claramente alguns pontos que abrangem diretamente a área de informática - Altos custos de implementação - Tecnologia não integrada – Tecnologia obsoleta, cara e não adaptável (*legacy*) – Alta latência de dados. Esta pesquisa mostra como a alta complexidade das arquiteturas financeiras atuais prejudicam todos os participantes do mercado. Legitimando o propósito deste trabalho. As tendências arquiteturais visam combater estas barreiras, usando o estado da arte em técnicas de interoperabilidade, adaptabilidade e integração.

A indústria de serviços financeiros está experimentando uma mudança arquitetônica para Services-Oriented Architecture (SOA), serviços Web e virtualização de recursos de TI. SOA tira partido do aumento de velocidade da rede e possibilita a virtualização e acoplação dinâmica de componentes de software. Isso permite a criação de novas aplicações sem perder o investimento em sistemas existentes de infraestrutura. O conceito tem o potencial de revolucionar a forma como é feita a integração, permitindo reduções significativas na complexidade e custo de tal integração (GigaSpaces 2004).

Para arquiteturas financeiras, a empresa Cisco Systems aposta em tecnologias como InfiniBand, RDMA e High-speed messaging bus (Cisco 2008). O barramento de mensagens deve usar um mecanismo confiável para entregá-las. Um conceito importante na distribuição de mensagem é o *stream* por tópicos, subconjunto dos dados do mercado definido por critérios tais como: ativo financeiro, bolsa de valores ou instrumentos financeiros. Subscrever em um tópico garante ao assinante o recebimento apenas da informação relevante ao tópico. No passado, todos os comerciantes recebiam todos os dados do mercado. Nos volumes atuais de tráfego, esta técnica seria sub-ótima.

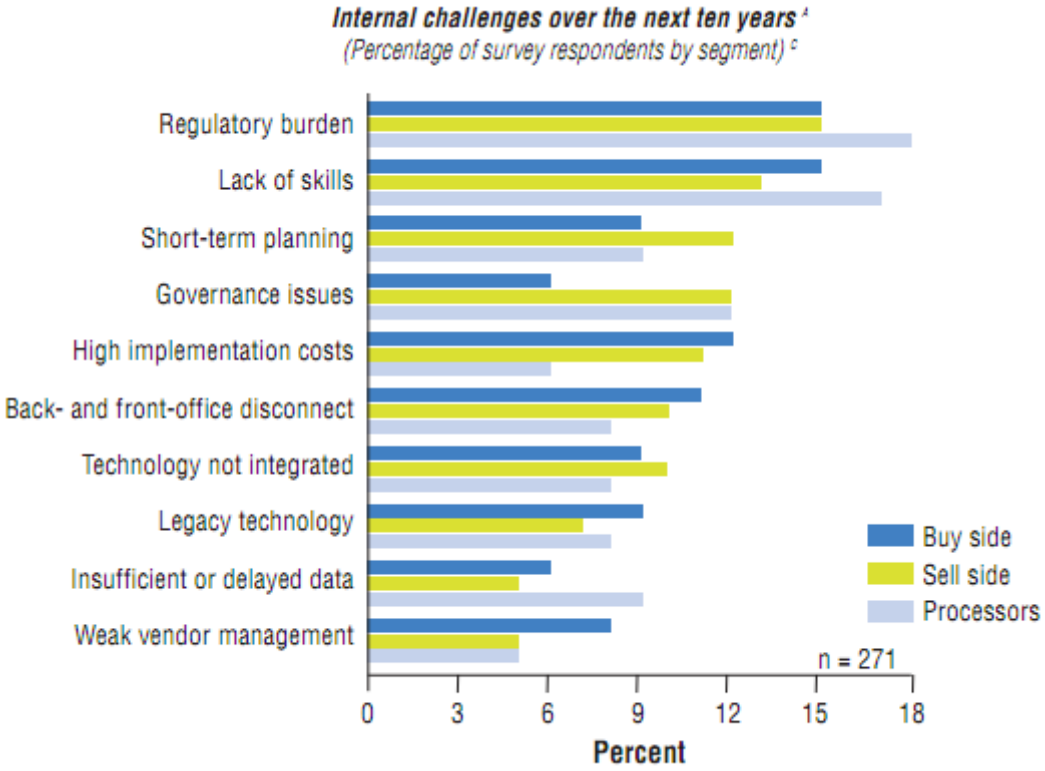

Figura 1.1: IBM Financial Markets Bussines Survey (IBM 2006)

Para o transporte de informação financeira os padrões FIX[2] (Financial Information Exchange) e FAST (FIX Adapted for Streaming) são largamente utilizados. Entretanto nota-se esforços para padronizar o barramento de mensagens. O protocolo aberto Advanced Message Queueing (AMQP) (Vinoski 2006) é defendido pela J.P. Morgan Chase e apoiado por diversas empresas de infraestrutura Cisco, RedHat, Microsoft, etc... Em arquiteturas como implementadas pela Eurex[3], uma das principais bolsas de derivativos do mundo, AMQP foi escolhido para o transporte interno de um ambiente Cloud como também para o transporte de informações financeiras, FIXML (FIX + XML) sobre AMQP (Klein 2011). A tendência SOA + AMQP não está apenas presente na área financeira, vemos esta vanguarda em projetos com requisitos semelhantes - alto desempenho e confiabilidade (CERN[4] e Nebula[5]).

1.3 Introdução a precificação de opções

A opção é um tipo de derivativo amplamente usado nos mercados, é um contrato que garante os direitos de compra ou venda de um ativo com preços e prazos de exercício preestabelecidos. Avaliar o preço adequado de uma opção exige que o investidor faça uma previsão do valor do ativo correspondente em uma data futura, o que pode-se, de forma grosseira, dizer que esta análise exige "prever o futuro dos preços". Independentemente da área, a previsão é uma tarefa complexa. Na economia a análise fundamentalista do

[2]http://www.fixprotocol.org/
[3]http://www.eurexchange.com
[4]European Organization for Nuclear Research
[5]Nebula Cloud Computing Platform - NASA

mercado, que tem como paradigma estudar a micro e macroeconomia, tem um certo grau de subjetividade em suas previsões, pois está baseada em interpretações. A análise técnica da economia abriu espaço para métodos estatísticos e probabilísticos no processo de previsão econômica.

Em 1973, Robert C. Merton publicou um artigo apresentando um modelo matemático que poderia ser usado para calcular um preço racional no negócio de opções (R.C Merton 1973). Este modelo foi chamado Black-Scholes e gera uma equação diferencial parcial de segunda ordem como solução. Como alternativa a esta equação diferencial P.P. Boyle propôs uma método numérico de avaliação usando o método Monte Carlo (Boyle 1977). O método Método Carlo é usado em diversas aplicações, especialmente simulações de ambientes ou sistemas onde coexistem diversas variáveis, graus de liberdade, e resolução de integrais. Seu algoritmo consiste basicamente em criar diversas amostras randômicas do problema e mesurar um resultado mediano das amostras. P.P. Boyle demonstrou que gerando-se diversos caminhos randômico dos preços do ativo até a data de exercício podemos encontrar o valor esperado da opção.

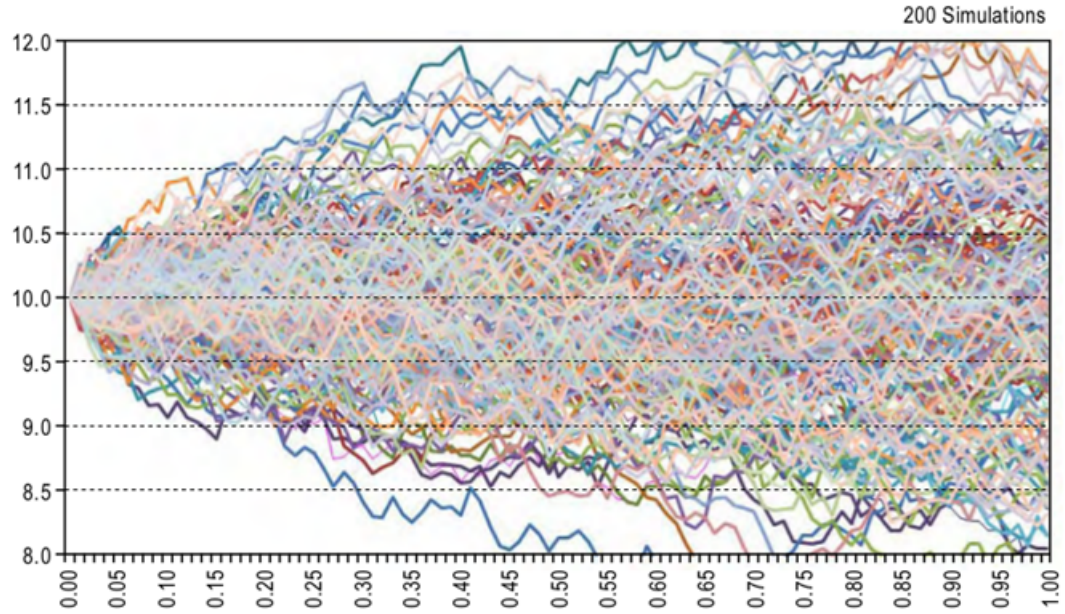

Figura 1.2: Simulações usando o modelo Black-Scholes

Apesar da simplicidade do método Monte Carlo, ele é classificado como um processo *CPU-Intensive*, pois para entradas pequenas podemos ter centenas de milhares de iterações, necessárias para uma previsão confiável. Para atender o requisito de baixa latência exigido pelo mercado, necessitamos que nosso sistema tenha um alto poder computacional. Atualmente as soluções mais velozes são implementadas em GPGPUs (Niramarnsakul e Chongstitvatana 2011) e FPGA (Schryver et al. 2011), mas o alto custo, consequente da complexidade de desenvolvimento e implantação destes produtos, acaba tornando-os acessíveis somente a empresas com alto poder aquisitivo. Razões como estas fazem com que produtos financeiros complexos necessitem de uma alternativa barata e confiável para mapear seu processamento em arquiteturas *many-core* de alto desempenho (Smelyanskiy 2008), como *grids* e *clouds* (Keetha e He 2009).

1.4 Objetivo

Este trabalho visa criar um *framework* acessível e flexível para distribuir produtos financeiros de alto desempenho e testá-lo em uma aplicação real, precificação de opções pelo método Monte Carlo. Para atender a baixa latência exigida pelo mercado, o *framework* utiliza troca de mensagens assíncronas em uma arquitetura distribuída orientada a serviços-eventos. A flexibilidade e acessibilidade são garantidas com padrões de interoperabilidade e integração. Além de um *framework* para paralelização de alto processamento, este projeto abrange serviços como: obtenção de dados do mercado em tempo real, interface com plataforma de negócio e visualização de resultados.

O modelo combina SOA, Event-Driven Architecture (EDA) e virtualização de máquinas para criar uma arquitetura similar a um *Cloud, IaaS*. Os serviços são iniciados remotamente e publicados em um servidor de mensagens, paradigma caixa postal. Aplicando o Message-Oriented Middleware (MOM), AMQP, que age como um intermediador de mensagens permitindo que componentes distribuídos se comuniquem com baixíssimo acoplamento por meio de streams e tópicos de mensagens, garantindo o fluxo dos dados e o discernimento de cada serviço.

A integração de diversos componentes distribuídos exige um esforço na definição das interfaces e fluxo de dados. Para contornar essas adversidades, padrões como serialização XML, Component-Based Development (CBD), Publish-Subscribe Channel (Hohpe e Woolf 2003) serão abordados. Para auxiliar na virtualização de máquinas (inicialização, monitoramento e gerenciamento de componentes distribuídos) foi desenvolvido o ProcessStarter, podendo este modelo ser usado para qualquer aplicação distribuída em uma rede com um servidor AMQP.

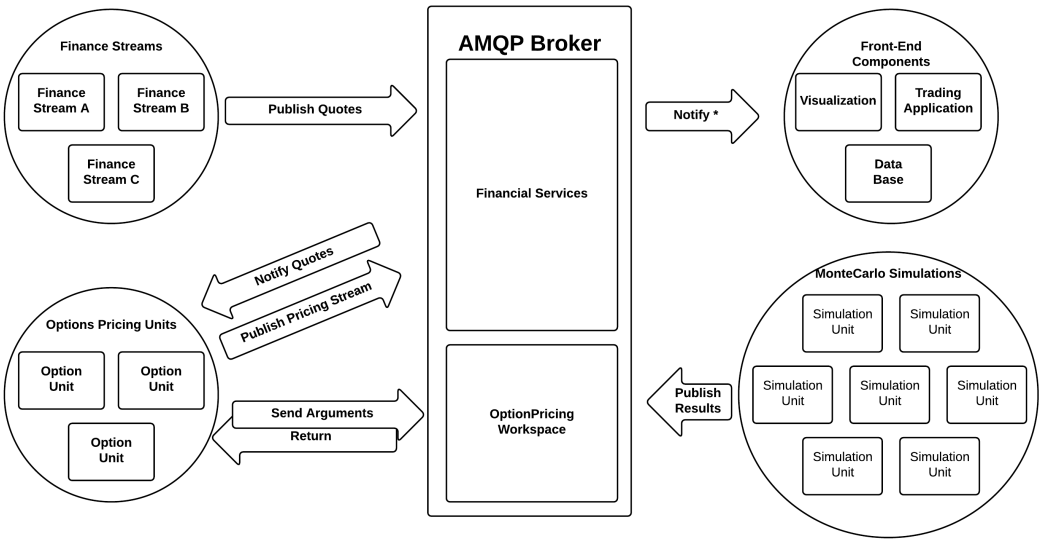

Figura 1.3: Modelo do Framework com precificação de opções

Seguindo o modelo de (Keetha e He 2009), sucintamente este projeto tem os seguintes objetivos:

- Criar um *framework* genérico para distribuir cálculos e serviços financeiros.
- Usar o *framework* para acelerar a precificação de opções; e avaliar seu desempenho.

- Aumentando a razão de mensagens por segundo, estudar os limites de processamento em componentes que usam comunicação via AMQP, serialização XML e paradigmas SOA-EDA.

2 PARADIGMAS PROPOSTOS

Neste capítulo serão introduzidos os conceitos que servirão de base para o modelo proposto no Capítulo 3, um *framework* para facilitar a criação de sistemas distribuídos na área financeira.

2.1 Padrões de integração e interoperabilidade

Enquanto a capacidade de processamento, armazenamento e dados gerados crescem em velocidades paralelas a Lei de Moore, já temos previsões de uma lacuna abrindo-se entre a grande demanda e pequena oferta de software. Países como a Estônia, que apontam a inovação tecnológica como fundamental para o desenvolvimento de sua economia, já iniciaram políticas de fomento ao desenvolvimento de software nas escolas de ensino médio, preparando uma nova geração de desenvolvedores para próxima era da humanidade, computação ubíqua, a onipresença da informática no cotidiano das pessoas.

A baixa qualidade de software, despreparo técnico dos desenvolvedores ou a falta de padrões e rigor no desenvolvimento são algumas evidências para deduzir esta baixa oferta. Certamente estes são fatores que afetam o mercado, mas o verdadeiro gargalo da pequena oferta de software está na integração de software. Hoje são poucos os sistemas criados com capacidade de interagir com ambientes externos. A possibilidade de integração de software evita a redundância de esforços, compartilhamento de recursos, um ganho maior que o reuso de código. Líderes de mercado investem profundamente na integração de seus aplicativos, repositórios disponíveis para a integração são encontrados facilmente entre esses líderes: Google Developers[1], Facebook Developers[2], Apple Developer[3].

Em arquiteturas orientadas a objeto, para desenvolver uma funcionalidade pode tornar-se necessária a compreensão de todo um sistema com diversos níveis hierárquicos, interfaces, sobrescritas, sobrecargas, etc... a falha na compreensão desta dinâmica do sistema facilita a inserção de erros e falhas no sistema, o que consequentemente aumenta o tempo necessário de análise, projeto, desenvolvimento e manutenção. Uma classe altamente reusável em um projeto orientado a objetos dificilmente é interoperável a outros sistemas pois tem uma série de vínculos indiretos (linguagem, arquitetura, etc...). Estas complexidades geradas na orientação a objetos dificultam a integração com sistemas externos.

Os métodos ágeis (Krogdahl e Luef 2005), a grande tendência no desenvolvimento de software, incorporaram arquiteturas SOA e EDA como portfólio. Estas arquiteturas

[1] https://developers.google.com/
[2] http://developers.facebook.com/
[3] https://developer.apple.com/

exigem um menor esforço entre projetistas e desenvolvedores, projeto e implementação estão desacoplados, não é necessária a compreensão de todo o sistema para a implementação do atendimento a um serviço ou um evento. Estas arquiteturas partem do princípio da criação de unidades com funcionalidades simples e auto contidas. Pode ser visto como algo prejudicial ao desenvolvedor retirar dele a onipresença sobre o sistema e vinculá-lo apenas a algumas funcionalidades, mas este reducionismo tem efeito contrário pois aumenta a produtividade e, consequentemente, a interação e satisfação dos participantes no projeto.

As arquiteturas como EDA e SOA aplicam diversos padrões de interoperabilidade para garantir a modularidade de componentes em relação ao sistema em que ele se encaixa. Essenciais a integração são interoperabilidade e reusabilidade, características intrinsecamente ligadas - um componente é 100% reusável somente se ele for 100% interoperável.

Visando a maximização da capacidade de integração do ambiente a ser desenvolvido neste trabalho, foram escolhidos paradigmas que visam principalmente alcançar interoperabilidade de maneira a reduzir seus efeitos adversos na latência do sistema. Estes padrões e paradigmas complementam-se no modelo final do trabalho, são eles:

- Component Based-Development (CBD)
- Message-Oriented Middleware (MOM)
- Serialização de objetos
- Arquiteturas orientadas a eventos (EDA) e serviços (SOA)

2.2 Component Based-Development

CBD popularizou-se na década de 90 quando a comunidade científica reconheceu a necessidade de um melhor gerenciamento sobre a complexidade de simulações multidisciplinares e um melhor endereçamento dos problemas de desempenho em arquiteturas paralelas e distribuídas (Armstrong et al. 1999). A proposta é um modelo de desenvolvimento mais flexível do que os oferecidos pela programação convencional baseado em chamadas sucessivas de sub-rotinas. CBD é baseado no encapsulamento de funcionalidades em componentes com interfaces bem definidas.

CBD foi considerado a evolução da orientação a objetos (OO), mas na época existiam diversos entraves para sua utilização. As principais dificuldades apresentadas por (Armstrong et al. 1999) são: (1) incompatibilidade de linguagens; (2) falta de padronização em comunicações entre objetos; (3) necessidade de compilação conjunta de interfaces. Este artigo cita diversas tecnologias para amenização destas dificuldades, entre elas: CORBA (OMG 1995), MPI (MPI 1995) e COM+ (Sessions 1997).

Apesar deste falso impulso inicial, o paradigma está novamente ganhando força com as diversas inovações tecnológicas tanto na comunicação quanto no desenvolvimento. Da necessidade de paralelizar o trabalho de desenvolvimento ágil e criar arquiteturas desacopladas, como SOA e EDA (Kajko-Mattsson e Lewis 2007). CBD reaparece em oposição a RUP e OOD (Crnkovic e Chaudron 2006).

2.3 Message Oriented Middleware

Sistemas MOM fornecem um modelo distribuído para comunicação assíncrona, este modelo não-bloqueante resolve muitas das limitações encontradas em comunicações ba-

seadas em sub-rotinas, RPC (Curry 2004). Os participantes de um sistema baseado em MOM não são bloqueados na entrega de mensagens, eles são autorizados a continuar o processamento, uma vez que uma mensagem foi enviada. Isso permite o envio de mensagens quando o remetente ou o destinatário está inativo ou indisponível para reagir em tempo de execução.

Em muitos sistemas os dados são distribuídos pelos componentes pelo uso de *polling*, essa técnica consiste em perguntar periodicamente ao servidor se existem novos dados, Figura 2.1. Em um sistema que a latência deve ser mínima, o *polling* é usado ostensivamente, causando uma sobrecarga desnecessária na rede. O padrão de mensagens *publish-subscribe* se tornou popular pois evita o *polling*, o servidor inicia o envio de dados recentes aos clientes inscritos em sua feed.

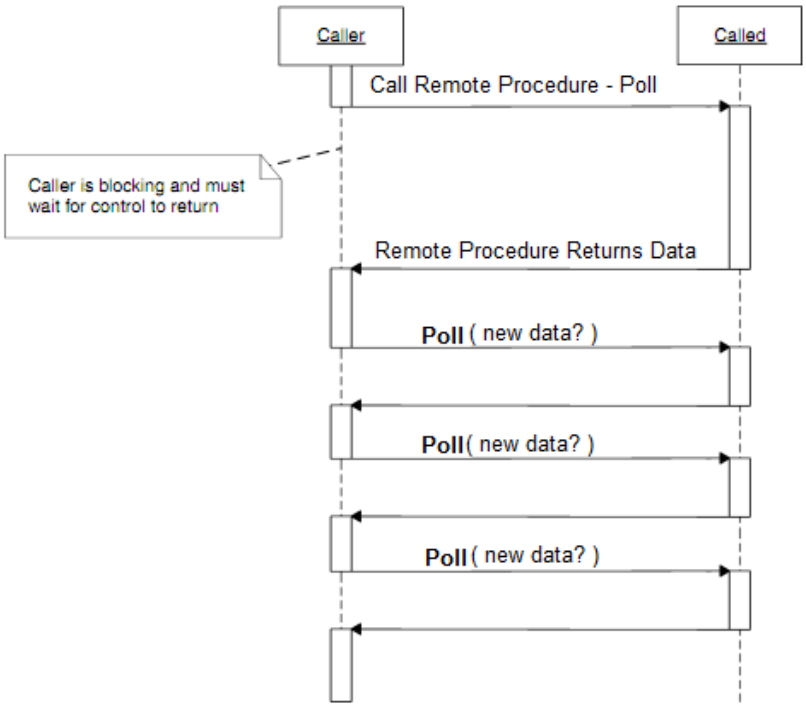

Figura 2.1: Diagrama sequencial de *polling* por RPC

Do padrão *publish-subscribe* surgiu o WebSockets, que se destina a clientes em aplicações Web. Com a popularização dos *clouds* tem se dado mais atenção a modelos de comunicação escaláveis. Neste contexto os message-oriented middlewares (MOM) estão crescendo pois garantem desacoplamento lógico, físico e temporal. Em sistemas financeiros MOMs tem se popularizado pois garantem a entrega com segurança e confiabilidade usando *broadcasting* e *multicasting*, vitais para a distribuição das cotações. A Figura 2.2 esquematiza a comunicação assíncrona de duas aplicações fazendo uso de um MOM.

Implantações de sistemas distribuídos baseados em MOM, como mostrado na Figura 2.3, é uma abordagem para comunicação de serviços entre diversas aplicações. A entrega de mensagens pelo modelo MOM é semelhante ao serviço postal. As mensagens são entregues aos correios, após, o serviço postal tem a responsabilidade pela entrega segura da mensagem (Tanenbaum e Steen 2002).

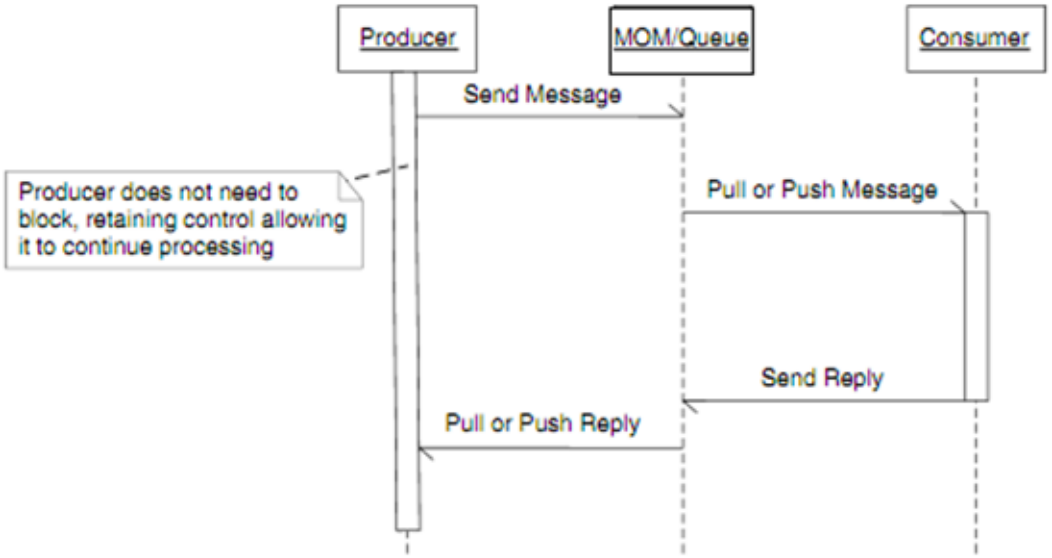

Figura 2.2: Modelo de comunicação assíncrona (Curry 2004)

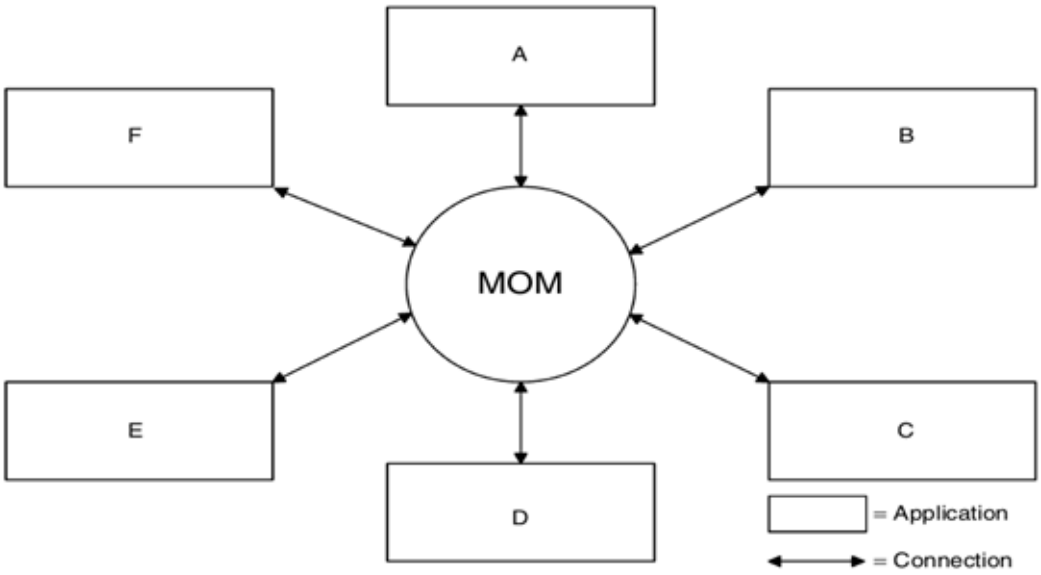

Figura 2.3: Diversas aplicações comunicando-se via message-oriented middleware (Curry 2004)

2.3.1 Advanced Message Queueing Protocol

AMQP é um protocolo de aplicação de padrão aberto pela OASIS[4] para MOMs. As definições do AMQP abrangem: orientação a mensagens, queuing, roteamento, confiabilidade e segurança (O'Hara 2007).

AMQP apenas define os comportamentos do servidor de mensagens e clientes, as diferentes implementações do protocolo de diversos fornecedores são totalmente interoperáveis. Pois assim como HTTP, FTP, SMTP, etc... o protocolo AMQP não define plataforma, linguagem de programação ou formato de dado. Diferentemente dos MOMs predecessores, como JMS, AMQP é um *wire-level protocol*, ou seja, o formato é enviado

[4]https://www.oasis-open.org/

pela rede através de um *stream* de octetos podendo ser interpretado por qualquer ferramenta que reconheça o protocolo. A Figura 2.4 esquematiza as camadas de rede AMQP sobre a camada de transporte.

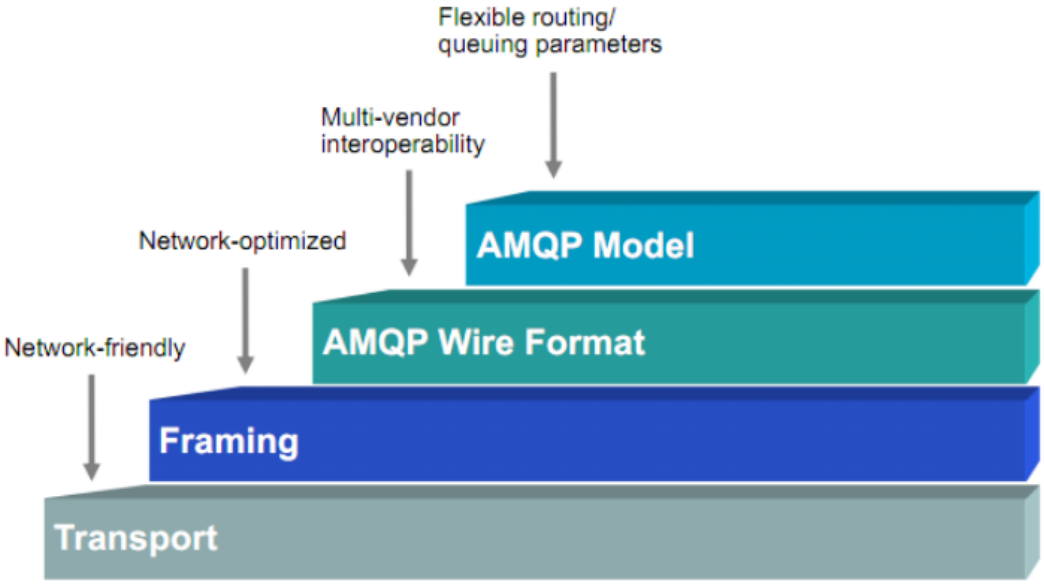

Figura 2.4: Modelo de camadas do protocolo AMQP

Além das tecnologias comuns oferecidas por MOMs (*point-to-point*, *publish-subscribe*, mensagens persistentes, transações, ACK, TTL, etc), AMQP cria o modelo de *queue*, *exchange*, *binding* e *topic*. *Queues* são *buffers*, de mensagens instanciados no servidor de mensagens. Uma *queue* pode ter vários produtores e vários consumidores, as mensagens são consumidas de acordo com a ordem de entrada, FIFO. As mensagens são consumidas por *round-robin* (uniformemente distribuídas entre os consumidores) ou por disponibilidade de consumo. O modelo da Figura 2.5 mostra a comunicação usando *queues* ponto a ponto e muitos produtores/consumidores por *round-robin*.

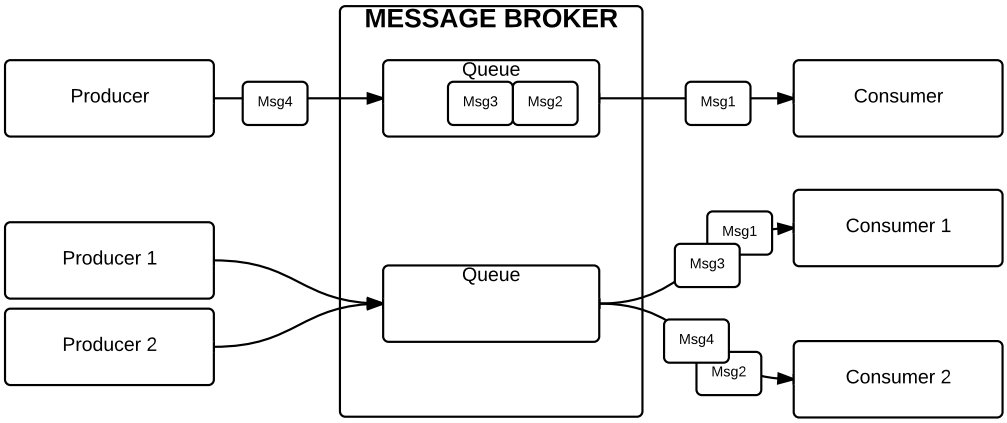

Figura 2.5: Comunicação ponto a ponto via AMQP

Uma *queue* pode estar associada a um *exchange*, por *binding*. O objetivo de um *exchange* é receber mensagens dos produtores e enviar estas mensagens para as *queues* de

acordo com o *bindign*. Cada *biding* pode ter um ou mais tópicos associados. Este modelo *queue-biding-exchange* funciona como o padrão *publish-subscribe*, todas as mensagens publicadas no *exchange* são enviadas para as *queues* subscritas, porém o *biding* filtra/seleciona apenas as mensagens relevantes ao tópico.

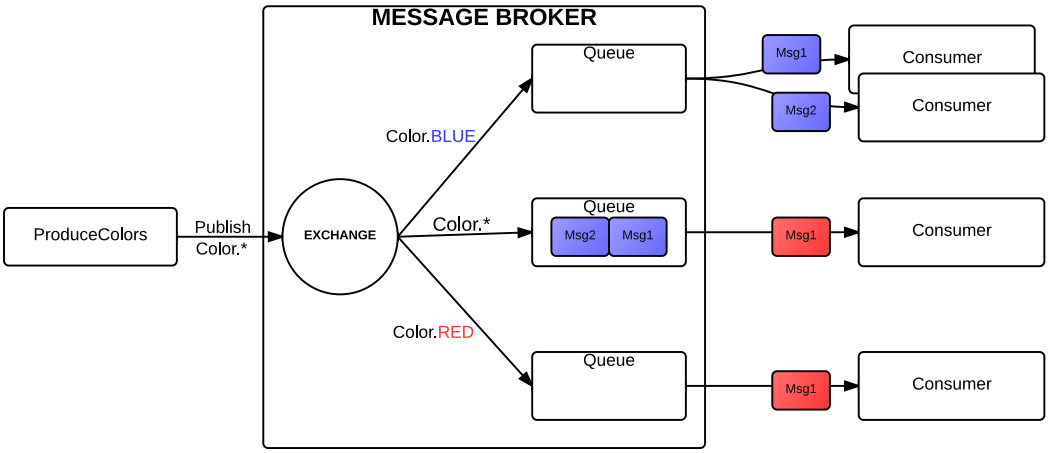

Figura 2.6: Comunicação por *exchanges* e tópicos via AMQP

Na Figura 2.6, é exemplificado o modelo de *exchange* com tópicos. O produtor publica no *exchange* mensagens com tópicos: *Color.RED* e *Color.BLUE*. Em seguida o *exchange* envia para as *queues* as mensagens relevantes a cada *biding*. A *queue* com *biding Color.** recebe todos as mensagens relativas a *Color*. Este modelo diminui a banda e processamento na rede pois diferentemente do padrão *publish-subscribe* ele evita que consumidores recebam mensagens irrelevantes ao seu serviço. Exemplo de tópicos: (1) stockexchange.NASDAQ.quote.*, recebe todas as cotações da NASDAQ; (2) stockexchange.NASDAQ.quote.technology, recebe as cotações de empresas tecnológicas com ações na NASDAQ, como GOOG(Google), FB(Facebook), etc.. (3) stockexchange.NASDAQ.quote.GOOG, recebe apenas as cotações da Google.

2.4 Serialização de objetos

Hoje, quando o desenvolvedor deseja criar uma comunicação entre componentes de diferentes linguagens de programação, ele enfrenta o obstáculo da incompatibilidade de dados ou formatos. As linguagens têm tipos bem definidos (*int, float, double, strings,* etc) que podem ser interoperáveis. Mas para o intercâmbio de estruturas definidas em linguagens não existe uma função padrão. O intercâmbio dos dados de uma estrutura definida em duas linguagens pode ser implementada facilmente por um programador experiente, mas esta implementação é referente apenas a estrutura definida e ao par de linguagens. Uma maior abstração desta função pode ser alcançado através de tecnologias como banco de dados, CORBA, SOAP-WebServices, etc... mas modelos como estes aumentam a complexidade da aplicação final e são dependentes de comunicação, plataforma, linguagem, etc.

O padrão de serialização é bem conhecido e muito usado. Serialização é o processo de converter o estado de um objeto em uma forma que pode ser mantido ou transportado. O complemento de serialização é a desserialização, que converte um *stream*, fluxo de dados, em um objeto. Em conjunto, estes processos permitem que os dados sejam facilmente

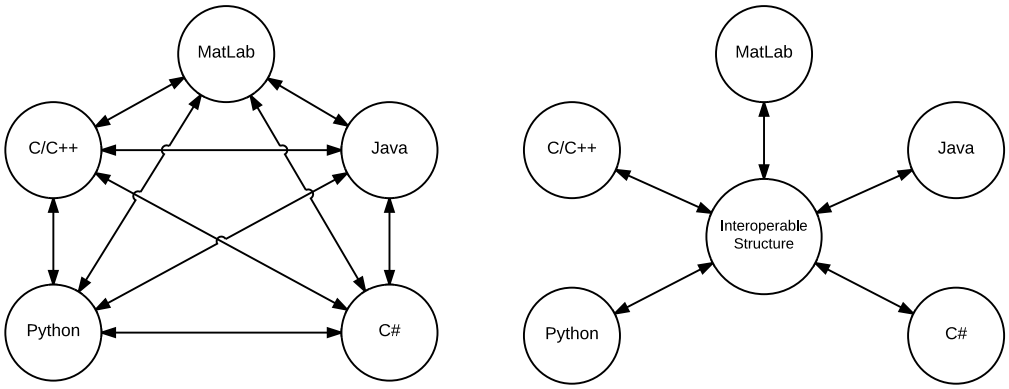

Figura 2.7: Serialização de dados, uma alternativa simples de interoperabilidade

armazenados e transferidos (Microsoft 2012).

Do padrão de serialização, livre de contexto, surgiram os padrões de formato XML e JSON. Mas por falta de esforço da comunidade, a serialização nativa das linguagens não suporta um padrão interoperável. Desta falta de padronização da estrutura surgiram diversas implementações interoperáveis. Mas novamente por falta de esforço estas não abrangem diversas linguagens. Por fim, uma aplicação envolvendo a comunicação de diversos componentes de linguagens diferentes acaba tendo que aplicar diferentes padrões para cada canal entre duas linguagens.

A Figura 2.7 apresenta dois modelos - a total falta de padronização destoando a um modelo com apenas um padrão interoperável. Este modelo foi prototipado e é descrito na Seção 5.1.3.

2.5 Arquiteturas orientadas a eventos e serviços

EDA e SOA são arquiteturas que visam desacoplar os componentes de software. SOA visa modularizar os serviços. EDA visa modularizar o fluxo de dados ou eventos, sendo os serviços representados como o fluxo de dados sobre uma composição de componentes. A Figura 2.8 sintetiza bem a ideia de cada paradigma. Derivada destas arquiteturas surge a sua combinação, ED-SOA (Levina e Stantchev 2009) ou SOEDA (Wieland et al. 2009), que representa os serviços como um conjunto de eventos.

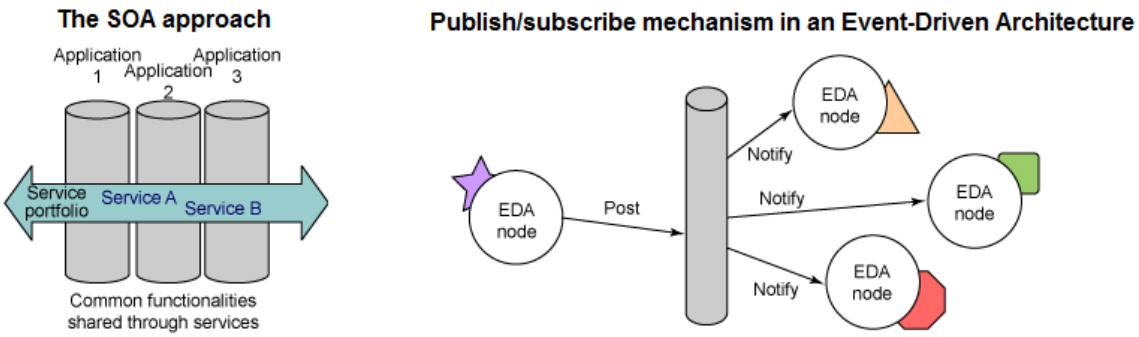

Figura 2.8: Arquiteturas SOA e EDA(Maréchaux 2006)

3 MODELOS E ARQUITETURA DO FRAMEWORK

3.1 Modelos

Nesta seção serão apresentados os modelos implementados no *framework*. A arquitetura faz uso extensivo destes modelos em sua composição .

3.1.1 Serviços publicados no servidor AMQP

Seguindo a metodologia SOA-EDA, 2.5, o modelo usa o servidor de mensagens AMQP para intermediar o fluxo de dados. Os fluxos de dados no servidor AMQP são representados como serviços. Componentes distribuídos publicam e subscrevem-se em serviços de maneira arbitrária. Quando um serviço é publicado o servidor AMQP o torna escalável, o números de componentes subscritos no serviço não é limitado. O esquema representado na Figura 3.1 fornece dois serviços A e B, publicados pelos respectivos componentes A e B, sendo B um processamento do serviço A.

As mensagens enviadas à um serviço não devem ser redundantes e sim a composição de diversos tópicos com produtos semelhantes. Exemplo: A.1, A.2 e A.3 representam cotações de ativos financeiros(1, 2 e 3) da bolsas de valores A. Assim usando o artifício de tópicos, um componente pode selecionar qual subconjunto do serviço A deve ser notificado. Exemplo: O componente B.1 necessita apenas as informações de A.1 para publicar o serviço B com tópico B.1, logo subscreve-se no serviço A com o subconjunto de tópicos A.1.

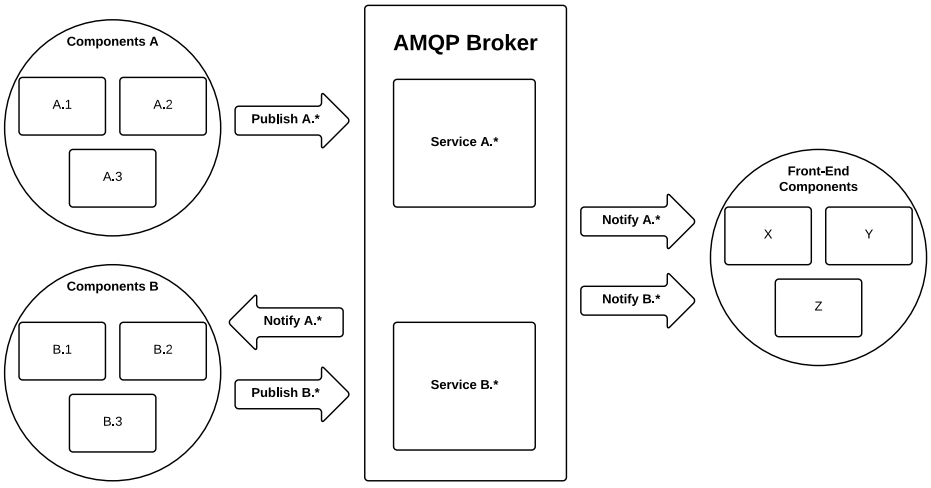

Figura 3.1: Modelo proposto de componentes distribuídos publicando serviços sobre AMQP

Quando um componente distribuído exerce um comportamento de subrotina (argumento e retorno, *RPC-like*) é criado uma *working queue* (cada mensagem argumento publicada é consumida por apenas um componente subscrito no serviço) ou *working publish* (a mensagem argumento publicada é notificadas para todos os componentes subscritos), dependendo da aplicação. Tendo um comportamento assíncrono, como descrito na Figura 2.2, estes casos podem necessitar de uma identificação para combinar a mensagem de argumentos e mensagem de retorno.

3.1.2 Componentes distribuídos

O modelo dos componentes distribuídos tem como princípio:

- Uso do paradigma CBD.
- A comunicação será estabelecida por AMQP.
- As interfaces serão definidas por um padrão de serialização.
- Sua funcionalidade está correspondente às arquiteturas SOA-EDA.

Devem ter uma funcionalidade fechada e executar como um processo independente. Podem ser implementados em qualquer linguagem e plataforma que suporte clientes do protocolo AMQP. Fica a cabo do projetista escolher quantas e quais streams subscritas e publicadas. Na definição das interfaces de cada *stream*, é aconselhado um formato de serialização que seja um padrão o mais interoperável possível, suportado por todas as linguagens da aplicação. As estruturas recebidas e enviadas não são necessariamente iguais. A Figura 3.2 mostra alguns padrões de componentes (Hohpe e Woolf 2003) e sua iteração com o servidor de mensagens AMQP.

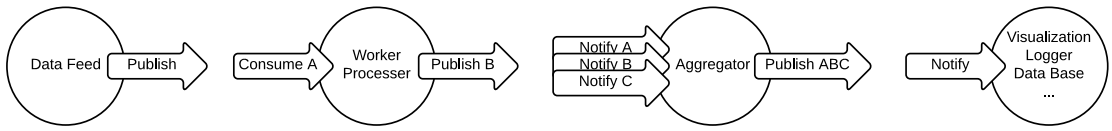

Figura 3.2: Padrões de componentes distribuídos.

A Figura 3.3 esquematiza a arquitetura de um componente interoperável, com um *stream* de entrada e um *stream* de saída. Note que a funcionalidade e formato/dados de entrada e saída são definidos em fase de projeto.

3.2 Arquitetura do framework

De acordo com os modelos apresentados anteriormente, nesta seção é definida a organização dos componentes do *framework*. Na Figura 3.4 vemos como diferentes classes de componentes interagem com o servidor AMQP, são elas:

- *Feeds* financeiros: São aqueles que fornecem as cotações e informações de bolsas de valores. Apenas publicam serviços de dados.
- Produtos financeiros: Tem funcionalidade implementada, processam e publicam serviços financeiros.

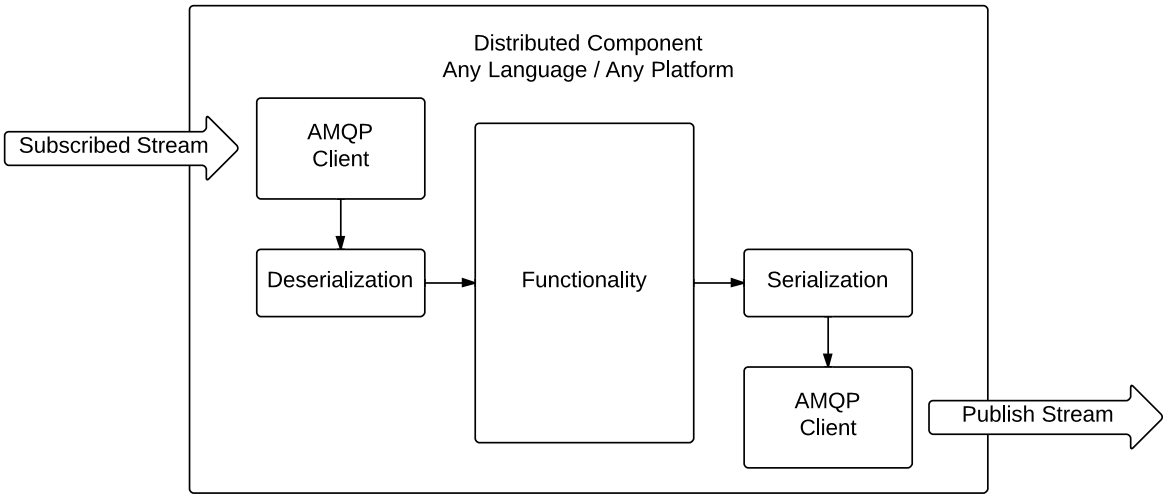

Figura 3.3: Arquitetura de um componente distribuído

- *Front-End*: Softwares de visualização, negócio ou armazenamento. São notificados pelos serviços desejados.

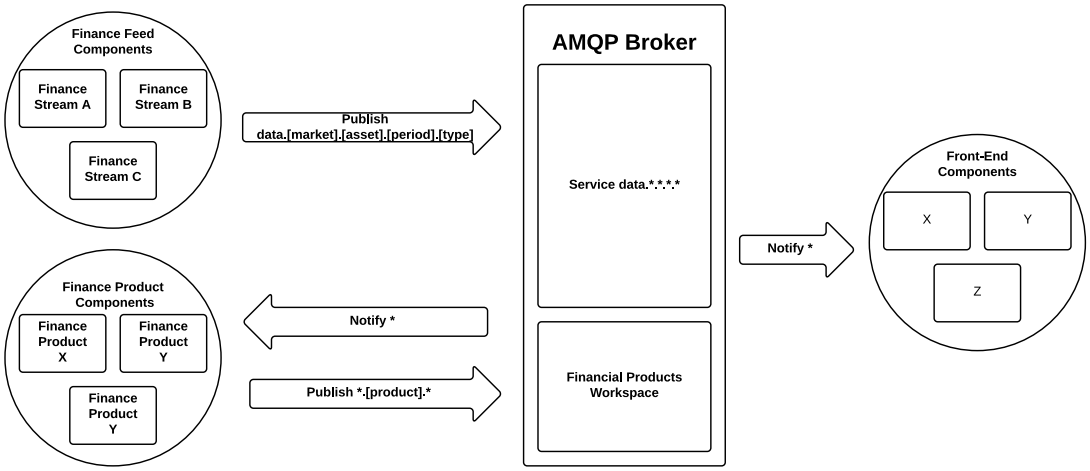

Figura 3.4: Arquitetura do *framework*

3.2.1 Feeds financeiros

Os dados de entrada de um sistema financeiro são as cotações (*quotes*) de ativos, representam variações no ativo, as cotações mais comuns são as que representam variações no preço do objeto de negócio. Neste *framework* as cotações vão ser classificadas por mercado, ativo, periodicidade e tipo. A publicação de cotações é feita no *exchange* chamado *datafeed* instanciado no servidor AMQP, e deve ter tópico com formato *data.[mercado].[ativo].[periodicidade].[tipo]*. As informações geradas por produtos financeiros pertinentes a um *data feed* ou a outras aplicações são publicadas no serviço de dados. A Figura 3.5 exemplifica feed financeiro do mercado FOREX, ativo EURUSD, periodicidade diária; O *feed* publica com o tipo *quote*(cotação), e o produto financeiro com o tipo *option* (opção sobre o ativo EURUSD).

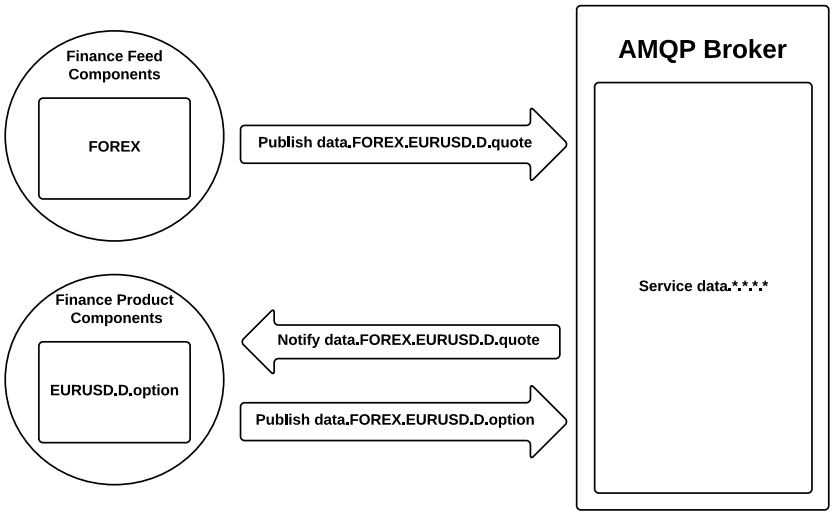

Figura 3.5: Exemplo de *feed* financeiro

3.2.2 Produtos financeiros e workspaces

Produtos financeiros necessitam da distribuição de seus cálculos entre componentes. A comunicação destes será feita através de um *workspace* no servidor AMQP, este *workspace* é constituído de um *exchange* instanciado no servidor AMQP, com nome arbitrário, exclusivo para as comunicações entre componentes deste produto. A Figura 3.6 demonstra um exemplo de *workspace* na avaliação de opções, que necessita a paralelização de suas simulações em componentes distribuídos. A notação de *workspace* também pode ser usada para a definição de serviços auxiliares como banco de dados ou virtualização, como criado no ProcessStarter, Seção 5.5.

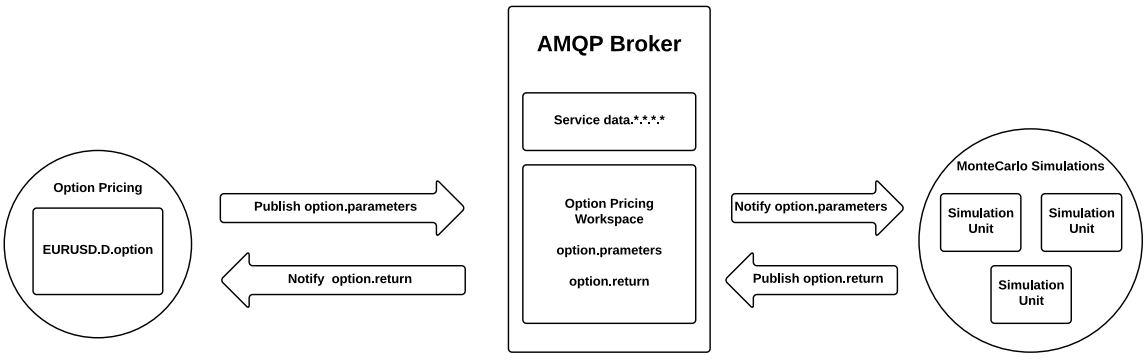

Figura 3.6: Comunicações entre componentes distribuídos através de *workspace*

4 AVALIAÇÃO DE OPÇÕES

Nesta seção será abordada a aplicação a ser implementada usando o *framework*. Abrangendo as definições básicas do mercado de opções, modelo e método de precificação e, por fim, o modelo de paralelização. A implementação em componentes distribuídos, conforme a arquitetura proposta na seção anterior, é descrita na Seção 5.4.

4.1 O mercado de opções

O Mercado de Opções é o mercado em que são negociados direitos de compra ou venda de um ativo, com preços e prazos de exercício preestabelecidos.

Esse mercado foi criado com o objetivo básico de oferecer um mecanismo de proteção ao mercado de ações contra possíveis perdas. Uma vez que os preços e retornos dos instrumentos financeiros estão sujeitos a flutuações imprevisíveis, as opções podem ser usadas para adaptar o risco às expectativas e metas do investidor. Os participantes do mercado que usam opções para limitar os riscos de oscilação de preços (operações de *hedge*) são conhecidos como *hedgers*. Entretanto, o mercado também precisa de participantes que estejam dispostos a assumir o risco: estes são chamados "especuladores".

As opções permitem que o investidor “alavanque” sua posição, aumentando o retorno potencial sobre um investimento sem aumentar o montante do capital investido, pois o capital investido inicialmente para comprar uma opção é relativamente pequeno em comparação com o ganho.

Contudo, quando dois investidores se comprometem em uma operação a ser realizada no futuro, os riscos são evidentes. Um dos investidores pode tentar cancelar a operação ou simplesmente pode não ser capaz de honrá-la financeiramente. Por esse motivo, todo capital aplicado em opções pode ser perdido, e o investidor (comprador) deve estar ciente desse risco. Por sua vez, o lançador de uma opção deve ter capacidade financeira para cobrir eventuais prejuízos potencialmente vultosos, bem como dispor de garantias suficientes para atender às exigências de margem (BM&FBOVESPA 2012).

4.2 O instrumento - opção

A opção é um instrumento financeiro que, como antes citado, garante a opção de compra ou venda sobre um ativo. Uma abstração deste instrumento fácil de compreender é uma opção de compra sobre uma casa: É criado um contrato entre o comprador e vendedor: Contrato de opção de compra sobre uma casa:

- Preço de Exercício (*strike*): Valor pré-determinado a ser pago pela casa.
- Data de Exercício: Data limite para o comprador decidir se vai realizar a compra.

Comprando este contrato é garantida a venda da casa pelo valor pré-determinado na data combinada. Se o mercado imobiliário super valorizar durante esse intervalo de tempo, o detentor do contrato escolhe a opção de exercer a compra, pois a casa está com um valor subestimado. Logo, terá lucro. Do contrário se estourar uma bolha imobiliária o comprador opta pela opção de não comprar a casa, minimizando suas perdas. A dificuldade do investimento com opções é estimar o valor que este contrato deve ser vendido, o prêmio, pois para isso exige a previsão dos preços no mercado imobiliário na data de exercício.

O Premio, isto é, o valor acrescentado ao contrato, que serve como compensação dada ao vendedor para garantir a exclusividade da opção de compra, pode ser calculado de diversas maneiras. Neste trabalho é apresentado o modelo matemático, Seção 4.4, e o método computacional, Seção 4.5, mais difundidos para alcançar um valor adequado.

Os contratos de opção de venda podem ser deduzidos facilmente, quem compra uma opção de venda, ao contrário da opção de compra, deseja vender o bem associado. Opções sobre operações de câmbio de moedas, *foreign-exchange options*, estão associadas a uma paridade cambial (eg. EUR/USD).

4.2.1 Tipos de opções:

Existem diversos tipos de opções, sendo as principais:

- Opção Americana: Uma opção do tipo americana possui um direito que pode ser exercido a um determinado preço a qualquer momento até a data de exercício.
- Opção Europeia: Uma opção do tipo europeia possui um direito que poderá ser exercido somente na data de exercício.
- Opção Asiática: Uma opção do tipo asiática o valor do prêmio é determinado conforme a média dos preços do ativo associado em um período presente.

As europeias são consideradas as Vanilla options, por serem consideradas as de mais fácil avaliação, nesta mesma faixa de complexidade encontramos a foreign-exchange option. Já as exóticas, com complexidade alta, tem em seu conjunto as asiáticas e variações das anteriores, como opções com barreira em que a opção é validada ou invalidada se atingir certo valor, também são chamadas de *path-dependents*, pois todos os valores intermediários até a data de expiração influenciam no preço.

4.3 Precificando opções

4.3.1 Valor Intrínseco

O valor intrínseco é a diferença entre o preço do ativo associado e o preço de *strike* da opção. Como o próprio nome diz, este valor já está agregado na opção. Em uma opção de compra se o preço de seu ativo associado for maior que seu preço de *strike*, dizemos que a opção está *in-the-money*, seu valor intrínseco é o valor do ativo menos o *strike*. Em uma opção de venda quando o valor do *strike* é maior que o valor do ativo, ela está *in-the-money*, e seu valor intrínseco é os valores de *strike* menos o ativo. Caso contrário o valor é zero, já que, por exemplo, não faz sentido exercer a opção de compra de um bem por um valor acima do oferecido no mercado aberto.

- Opção de compra:
  $IntrinsicValue = (AssetPrice > Strike)?AssetPrice - Strike : 0;$

- Opção de venda:
  $IntrinsicValue = (Strike > AssetPrice)?Strike - AssetPrice : 0;$

4.3.2 Valor extrínseco

O valor extrínseco, também chamado de valor temporal, pode ser encontrado subtraindo o preço da opção, o prêmio, pelo valor intrínseco, esta margem de valor representa o valor da aposta, quanto mais arriscada a posição da opção, maior o valor extrínseco. É a compensação ao risco monetário que o escritor/criador da opção fica submetido. Podemos citar alguns valores que influenciam no prêmio de uma opção:

- Preço do ativo associado: É o valor que tem mais impacto pois as flutuações no preço do ativo são diretamente proporcionais no preço, quando a opção está in-the-money.

- *Strike*: se o preço do ativo associado está perto do preço de *strike* da opção, mudanças no preço do ativo vão se expressar com mais força no valor de prêmio. Opções que o *strike* está distante do valor do ativo não sentem tanto a pressão especulativa.

- Data de expiração: Quanto mais perto da data de expiração da opção menor o valor extrínseco pois o risco diminui. A compra de uma opção que tem a data de expiração para o próximo ano tem um valor extrínseco alto, pois é exigido ao investidor a previsão do valor do ativo associado no ano seguinte.

- Volatilidade do ativo: Quanto mais volátil o ativo associado maior o valor extrínseco, pois é difícil prever o preço na data de expiração, quando o ativo apresenta mudanças bruscas de valor durante o período analisado.

Além das variáveis acima, podemos incluir no cálculo do prêmio da opção: taxa de fundos com risco livre, inflação ou um possível acumulados de lançamento futuros de dividendos do ativo. Por exemplo, no caso de uma opção de venda o vendedor poderia ter vendido suas ações por conta própria e aplicado o dinheiro em um fundo de renda fixa, certificado de depósito bancário (CDB) ou caderneta de poupança que lhe garantem um certo lucro com risco mínimo.

Como vimos, precificar uma opção é achar o valor adequado ao valor de prêmio, o preço adequado segue a meta de neutralizar o risco. Implicitamente, o investidor que tiver o melhor método de avaliação dentre seus concorrentes tem menor risco, pois tem uma base para especulação ou hedge mais segura, assim tende a reduzir as perdas e aumentar o lucro. Existem diversas formas de avaliar o preço, a mais difundida usa o modelo Black-Scholes.

4.4 Modelo Black-Scholes

Em 1973, Robert C. Merton publicou um artigo apresentando um modelo matemático que poderia ser usado para calcular um preço racional no negócio de opções. (Mais tarde ele recebeu um prêmio Nobel pelo seu trabalho) No mesmo ano, opções foram inicialmente negociados no mercado aberto. Desde então, a demanda por contratos de opções cresceu ao ponto de superar de longe os volume de negócio de seus ativos associados. Com as ideias do trabalho de Merton, dois outros pesquisadores, Fischer Black e Myron Scholes, criaram um modelo de precificação que tornou-se conhecido como o modelo Black-Scholes. O modelo Black-Scholes usa algumas premissas:

- Existe uma taxa de juros, livre de risco, constante e igual, tanto para empréstimo quanto para financiamento.
- O preço segue um movimento Browniano geométrico com tendência e volatilidade constantes.
- Não há custos nas transações.
- O ativo não paga dividendos.
- Não há restrições para a venda a descoberto.

Apesar destas premissas não estarem de total acordo com o mercado financeiro o modelo Black-Scholes consegue estimar valores de contrato de opções em um nível de neutralidade de risco suficientemente alto, tornando-se popular entre os investidores.

Existem novos modelos baseados ou expandidos do modelo Black-Scholes, alguns corrigem diferenças na cobrança da taxa de juros, outros corrigem o preço do pagamento de dividendos. Mas o modelo que se sobressaiu entre esses é o Heston, pois corrigiu o fato da volatilidade não ser constante. O modelo Heston assume que a volatilidade tem um comportamento estocástico, assim como os preços.

O modelo Black-Scholes permitiu obter fórmulas fechadas para a avaliação de opções, mas este modelo nos leva a equações diferenciais parciais de segunda ordem. Como alternativa a estas equações diferenciais P.P. Boyle propôs uma método numérico de avaliação (Boyle 1977), o método Monte Carlo.

4.5 Método Monte Carlo de precificação

4.5.1 Visão geral do método Monte Carlo

O método Monte Carlo é um método numérico usado em diversas aplicações, especialmente simulações de ambientes ou sistemas que coexistem diversas variáveis, graus de liberdade, e resolução de integrais. Seu algoritmo consiste basicamente em criar diversas amostras randômicas do problema para achar o resultado. Como o algoritmo depende de amostras, é necessário que essas sejam criadas da maneira mais randômica possível, pois caso contrário o resultado pode não convergir. Quanto melhor distribuídas as amostras geradas por um gerador de números pseudo randômicos, mais veracidade garantirá aos resultados do método Monte Carlo.

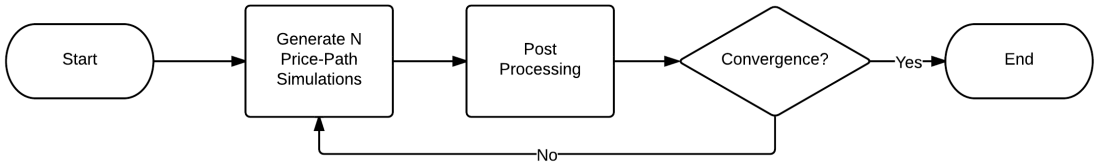

Figura 4.1: Fluxograma do método Monte Carlo.

Além do custo da criação dos números randômicos temos o custo da quantidade de amostras, validação das amostras e agregação dos valores das amostras em busca do resultado. Visto que esse algoritmo demanda um alto custo computacional ele é geralmente aplicado para casos em que o método trivial ou determinístico é inviável.

Um exemplo clássico do Método Monte Carlo é o problema da agulha de Buffon, 4.2, neste caso o naturalista francês Georges de Buffon montou um quadro com ripas de madeira paralelas e queria saber qual a probabilidade de uma agulha jogada dentro desse quadro pousara sobre as linhas que separam as ripas. Existe uma distribuição probabilística da queda da agulha no quadro, sendo este um dos primeiros problemas de geometria probabilística. A probabilidade da agulha cair nas linhas pode ser alcançada de duas maneiras: (1) Usando geometria integral. (2) Usando o método Monte Carlo: simulando a queda da agulha sucessivamente cria-se uma população, por fim com a população calcula a probabilidade da queda da agulha nas linhas estatisticamente.

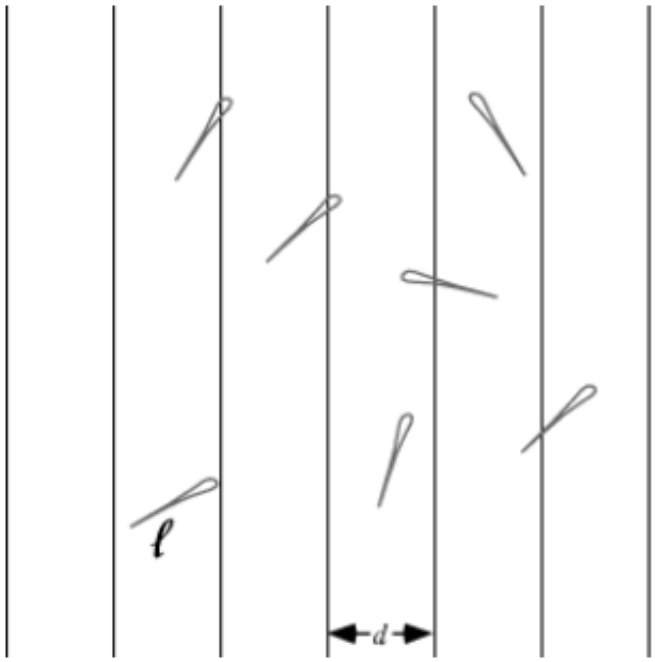

Figura 4.2: Experimento 'Agulhas de Buffon'

Como visto as amostras, simulações da queda da agulha, devem ser randômicas e independentes entre si para ter uma distribuição fiel da realidade, esta técnica só foi alcançada com o surgimento dos computadores e geradores de números randômicos. De fato o método Monte Carlo junto com o computador, tem suas origens no projeto Manhattan.

4.5.2 Resolvendo o modelo Black-Scholes

Em 1976, Cox e Ross analisaram o modelo Black-Scholes e propuseram uma abordagem diferente para a avaliação de opções. Eles mostraram que dado certos parâmetros do mercado, alguns citados na Seção 4.3.2, podemos calcular a distribuição esperada do preço do ativo associado na data de expiração da opção. Desta distribuição podemos obter o valor da opção na data de expiração à partir de uma integral. Em 1977 P.P. Boyle apresenta uma maneira muito simples e flexível, aos diversos tipos de opção, para resolver esta integral aplicando o método Monte Carlo. Sendo:

- $S(t)$, o preço do ativo subjacente na data t.
- δt, uma variação positiva arbitrária no tempo.
- r, a taxa de risco livre.

- σ, volatilidade do ativo subjacente.
- ϕ, uma variável randômica sobre a distribuição normal.

Podemos simular o preço de ativo, S, na data futura $(t + \delta t)$ pela fórmula:

$$S(t + \delta t) = S(t) * exp\{(r - \frac{1}{2}\sigma^2)\delta t + \sigma\phi\sqrt{\delta t}\}$$

Entretanto para calcularmos o preço da opção, são necessárias diversas amostras simuladas. Logo uma opção de compra do tipo Europeia com N amostras simuladas pode ser expressa como:

$$V(So, t = 0) = \frac{1}{n}\sum_{i=1}^{n} e^{-rT} * max[So * exp\{(r - \frac{1}{2}\sigma^2)T + \sigma\phi i\sqrt{\delta t}\} - E, 0]$$

Sendo E o preço de exercício, *strike* e $T = (t+\delta t)$ igual a data de expiração da opção. Quanto mais complexa a opção, maior deve ser o N. Para achar o erro padrão da fórmula acima, podemos usar estatística básica sobre os resultados, sendo ω o desvio padrão das simulações, o erro padrão pode ser encontrado:

$$\frac{\omega}{\sqrt{N}}$$

Assim sendo μ o preço médio, podemos construir o intervalo de confiança, o valor exato de V é dado por:

$$\mu - \psi\frac{\omega}{\sqrt{N}} < Ve < \mu + \psi\frac{\omega}{\sqrt{N}}$$

Aonde ψ determina o percentual de confiança. Vemos assim que para reduzir o erro em um fator de 2, devemos quadruplicar o número de simulações. Usando o método Monte Carlo convergimos para um resultado em $O(1/\sqrt{N})$ (Johnson 2008).

A solução encontrada por P.P Boyle além de resolver a integral de modo numérico, abriu espaço para a precificação de opções exóticas, pois podemos simular a evolução do mercado de modo iterativo. Para uma opção de compra Europeia, usamos $T = (t + \delta t)$, sendo $t = 0$, a data atual, e δt igual ao número de passos (dias) necessários para a alcançar a data de expiração da opção. Na opção Europeia básica, Vanilla, o prêmio só é influenciado pelo Ve encontrado e o *strike*. Mas em opções exótica em que os passos, o caminhar dos preços, influenciam no valor final da opção (*path-dependents*), é necessária a iteração de cada passo até a data de expiração, tendo de simular assim os preços de cada dia.

Esse método de gerar todos os valores intermediários até a data de expiração, aumenta a complexidade em δt (uma iteração para cada passo), para compensar esse aumento computacional existem métodos para acelerar a convergência do preço, diminuindo a variância (e.g Multi Level Monte Carlo). Estas técnicas não serão abordadas nesta aplicação.

4.5.2.1 Gerador de números randômicos

O método Monte Carlo necessita de milhares de amostras, cada amostra é a simulação dos passos do preço de um ativo pelo método Black-Scholes, cada passo simulado necessita como entrada um numero randômico sobre a distribuição normal, ϕ, é sabido que por métodos aritméticos não é possível gerar números verdadeiramente randômicos,

independentes entre si, mas existem técnicas matemáticas para refinar os resultados das amostras pseudorrandômicas. Neste trabalho foi implementado um gerador de números pseudorrandômicos com a técnica Transformada de Box-Muller (Box e Muller 1958), muito usada em simulações do método Monte Carlo pois tem uma velocidade e intervalo de confiança alto.

4.6 Modelo de paralelização

Antes da modelagem da distribuição do método Monte Carlo é sugerida a análise do seu fluxograma sequencial, Figura 4.1, e subsequente paralelização: (1) Gera-se N, um número arbitrário, de simulações. (2) Calcula-se o erro padrão e intervalo de confiança das amostras simuladas. (3) Decide-se um intervalo de confiança satisfatório; (sim) retorna o valor mediano e o intervalo de confiança; (não) volta ao passo 1. A etapa com maior carga de processamento é a etapa de geração das simulações, o método Monte Carlo e o modelo de Black-Scholes, esta pode ser paralelizada da como mostra a Figura 4.3.

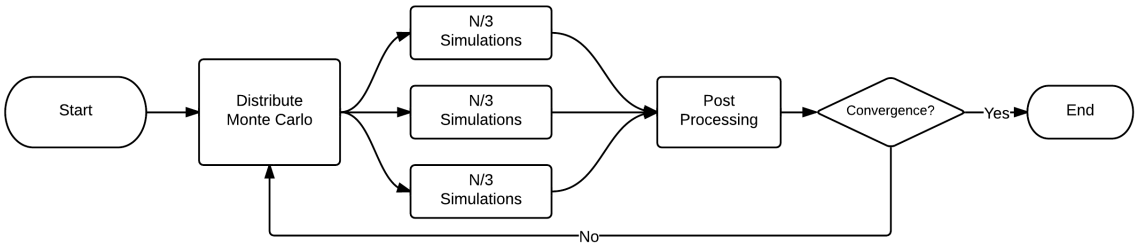

Figura 4.3: Método Monte Carlo paralelizado por três unidades.

A carga de simulações fica dividida entre as unidades de paralelização. No fluxograma da Figura 4.3 a carga é dividida por três. É inviável para ambientes distribuídos que todos os valores gerados durante a simulação sejam retornados a um nodo principal de pós processamento. Para diminuir essa carga sobre a comunicação, é feito um pré processamento das amostras em cada unidade de paralelização, após, são enviadas somente as informações pertinentes à unidade principal de pós-processamento, como mostrado na Figura 4.4.

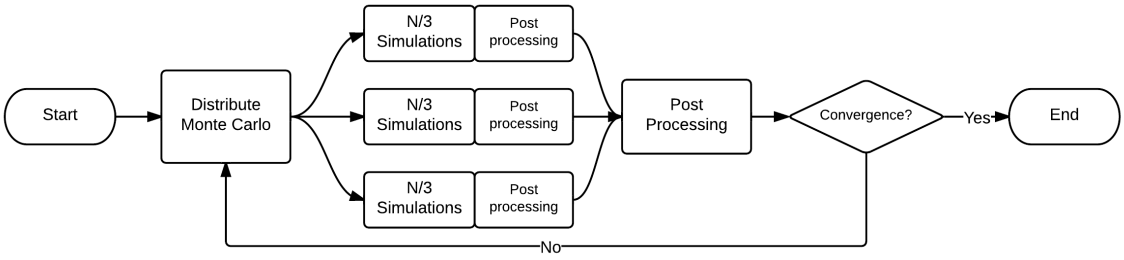

Figura 4.4: Modelo de paralelização com pré-processamento em cada unidade distribuída.

Sendo X o resultado de N amostras, $E(X)$ é o valor esperado de X. Como as amostras tem igual probabilidade, $E(X)$ é igual a média dos valores das amostras. Podemos calcular o desvio padrão da seguinte maneira:

$$\sigma = \sqrt{E((X - E(X))^2)} = \sqrt{E(X^2) - (E(X))^2} = \sqrt{1/n * \sum_{i=1}^{n}(Xi^2) - (1/n * \sum_{i=1}^{n} Xi)^2}$$

Logo para agregar os valores distribuídos necessitamos que cada unidade retorne $E(X^2)$ e $E(X)$. Para o fluxograma da Figura 4.4, com três unidades, temos a seguinte fórmula:

$$\sigma = \sqrt{(\frac{E1(X^2) + E2(X^2) + E3(X^2)}{3}) - (\frac{E1(X) + E2(X) + E3(X)}{3})^2}$$

Disparar as simulações em diferentes unidades paralelas e sincronizar os resultados para o pós-processamento tem um custo alto (e.g. chamada RPC síncrona), consequentemente adotou-se a técnica de simular uma quantidade de amostras suficientemente grande para sempre convergir a um intervalo de confiança satisfatório, ou seja, para não quebrar o fluxo aumenta-se o N e se retira a decisão 'Convergence?' do fluxograma na figura 4.4. Em uma comunicação assíncrona, podemos facilmente criar uma arquitetura que envia pacotes de tarefas aos nodos de simulação e agrega os resultados dinamicamente.

5 PROTOTIPAÇÃO

5.1 Tecnologias

Nesta seção serão introduzidas as tecnologias usadas no protótipo.

5.1.1 Servidor de mensagens AMQP – RabbitMQ

RabbitMQ é um servidor de mensagens, MOM, de código aberto que implementa o protocolo AMQP. RabbitMQ é escrito em Erlang e foi construído sobre o *framework* Open Telecom Platform, para clusterização e tolerância a falhas. O suporte e desenvolvimento são feitos pela Vmware. Os códigos fontes são disponibilizados sobre a Mozilla Public License. O projeto RabbitMQ consiste em:

- Servidor de mensagens RabbitMQ
- *Gateways* para os protocolos HTTP, STOMP e MQTT.
- Clientes AMQP para Java, .NET Framework e Erlang (clientes AMQP em outras linguagens são distribuídos por outros fornecedores).
- *Pulg-in* de gerenciamento, replicação de mensagens e federação de servidores.

A Figura 5.1 mostra todas as interfaces AMQP implementadas no produto RabbitMQ, criado pela VMware.

5.1.2 Feed e front-end financeiros - MetaTrader 5

MetaTrader 5[1] é uma plataforma de negócio eletrônico amplamente utilizado por investidores no mercado FOREX, mercado decentralizado de câmbio de moedas, Figura 5.2. Foi desenvolvido pelo MetaQuotes Software. O cliente é um aplicativo baseado no sistema operacional Microsoft Windows e tornou-se popular principalmente devido à capacidade dos usuários escrever seus próprios scripts que podem automatizar a negociação. Além dos componentes básicos de análise técnica, o terminal de clientes inclui um editor e um compilador com acesso a uma biblioteca colaborativa de software, documentação, artigos e ajuda. O software utiliza uma linguagem de script proprietária - MQL5, muito parecida com C++. Complementar ao cliente existe o componente servidor, este é executado pelo corretor fornecendo o fluxo de dados dos preços e serviços de corretagem.

Neste trabalho é usada a versão grátis demo do cliente, da MetaQuotes Software[2], que fornece uma conta para simular operações no mercado FOREX e os streams de dados

[1] www.metatrader5.com
[2] www.metaquotes.net

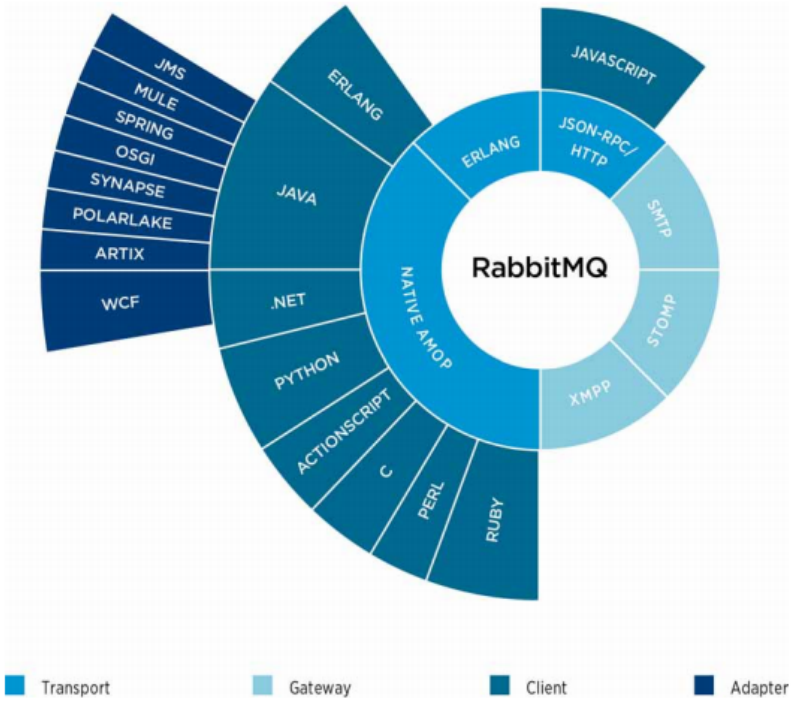

Figura 5.1: Abrangência de interfaces do RabbitMQ (RabbitMQ 2012)

Figura 5.2: MetaTrader 5

dos principais mercados FOREX, dólar dos Estados Unidos da América (USD) com suas paridades cambiais: EUR, GBP, CHF, JPY, CAD, AUD; em diferentes granularidades periódicas (1m, 5m, 10m, 30m, 1h, diário, ...). O estabelecimento da comunicação com o framewok e AMQP é dada por meio de DLLs implementadas em C#.

5.1.3 Serialização de objetos - XmlSerializer

Com o modelo proposto na Seção 2.4, implementou-se um serializador de objetos para XML interoperável entre múltiplas linguagens.

Com o largo uso de XML na troca de dados sobre a internet, a maioria das linguagens

de programação acabaram adotando parsers XML. Os parsers XML estão em bibliotecas nativas da linguagem ou são terceirizados, alguns de código aberto. Isso tornou o XML um padrão largamente aceito e interoperável, mas no contexto de troca de dados e objetos ainda persiste a falta de compatibilidade. Desenvolvedores ainda devem se preocupar com a conversão dos dados dentro do XML. A falta de uma solução simples e acessível com interoperabilidade entre diversas linguagens é a principal motivação deste serializador XML.

O XmlSerializer consiste em uma coleção de componentes, um para cada linguagem de programação. O padrão de serialização adotado é baseado no esquema simples (nomes associados a valores) encontrado na biblioteca nativa de C#. Os componentes XmlSerializer implementam as funções de serialização e deserialização. O XML é uma codificação "human-readable", fazendo com que a lógica de comutação de um dado estruturado para XML seja plenamente entendido. Por exemplo na Figura 5.3 é feita a comutação de um objeto C# e XML. A raiz do XML recebe o nome da classe C#, e aninhados na raiz os membros do objeto.

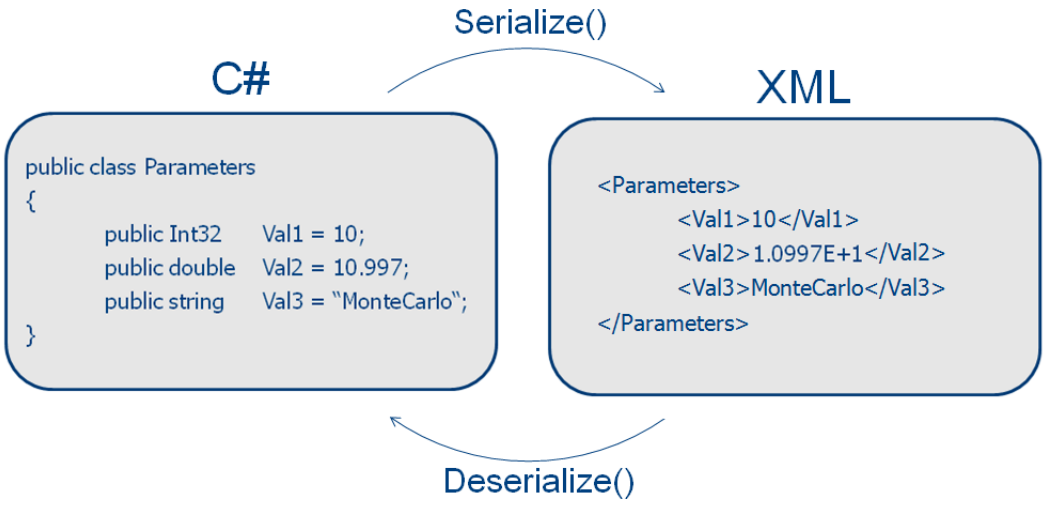

Figura 5.3: Comutação entre uma classe C# e uma estrutura XML

O XmlSerializer faz reuso de bibliotecas interoperáveis em C# (nativa) e Java (código aberto), e foi implementado para Python, C++ e MatLab; abrangendo as linguagens expostas no modelo da Figura 2.7. Este padrão é aberto e independente de comunicação, logo facilmente extensível em outras linguagens. Cada linguagem tem suas características, logo a definição dos objetos interoperáveis deve seguir um modelo de acordo com o XmlSerializer de cada linguagem. Na Figura 5.4 temos o câmbio via XML de uma estrutura de dados em C# e C++, note que a estrutura 'Parameters' é definida como class em C# e struct em C++.

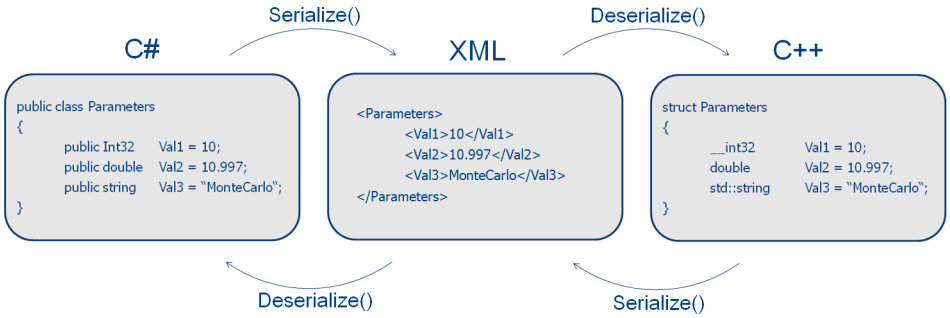

Figura 5.4: Serialização entre estruturas XML e estruturas nas linguagens C# e C++

5.2 Componentes distribuídos

Foram implementados em Java e Python classes e interfaces que transformam a conexão com o servidor AMQP e serialização XML transparentes ao desenvolvedor. Algumas das classes bases:

- ChannelsFactory: classe que usa o padrão *factory* para criar canais e conexão com o servidor AMQP.
- ReceiverMQ e SenderMQ: interfaces abstraem tipos diferentes de canais AMQP.
- SerializableClass<ClassType> : interface genérica de serialização e desserialização entre classe de tipo *ClassType* e *byte[]*.
- ReceiverThreaded<ClassType> SenderThreaded<ClassType>: Classes que agregam a serialização e comunicação AMQP em uma thread com buffer de mensagens.

```
1   ...
2   // instantiate the receiver buffer/thread with
3   // class type, serialization class and AMQP channel
4   ReceiverThreaded<SimulationReport> reportsReceiver =
5   new ReceiverThreaded(simulationReportXML,receiverMQ);
6
7   // start thread
8   reportsReceiver.Start();
9
10  // functionality
11  while(true)
12  {
13      SimulationReport report = reportsReceiver.Consume();
14      DoSomething(report);
15  }
16  ...
```

Figura 5.5: Exemplo de uso das classes e interfaces do framework

5.3 Ambiente financeiro

Usando as tecnologias mencionadas a arquitetura do ambiente financeiro usando o *framework* pode ser visualizada na Figura 5.6. O servidor de mensagens RabbitMQ possibilita intermediação das comunicações via AMQP e MetaTrader 5 cria o fluxo de dados financeiros do mercado FOREX, podendo este software ser usado para visualização e negócios automatizados.

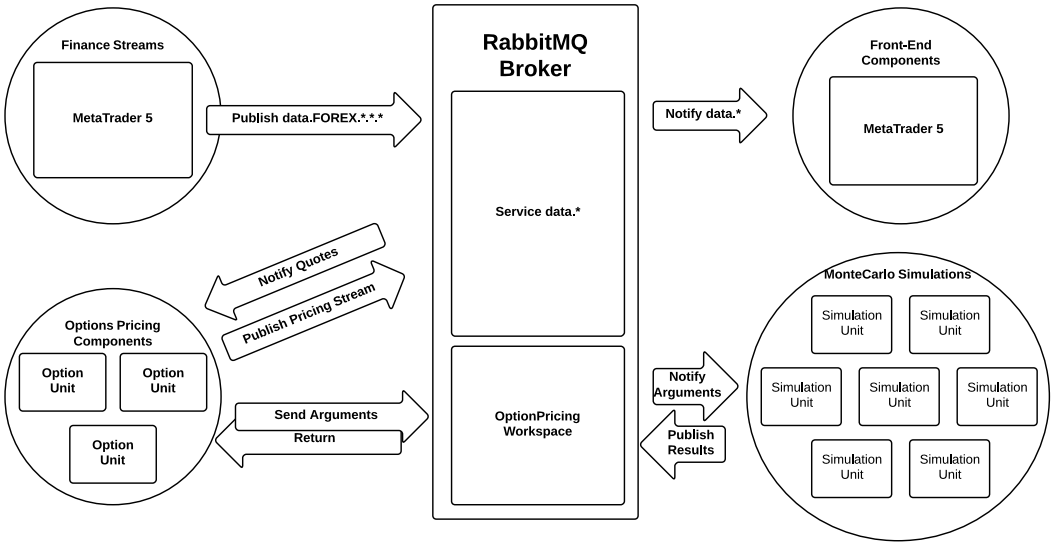

Figura 5.6: Prototipação do ambiente financeiro com avaliação de opções

5.4 Prototipação da avaliação de opções

A avaliação de opções em componentes distribuídos foi implementada em Java e está dividida em duas classes de componentes:

- PricingUnit – responsável pela precificação de uma opção.
- SimulationUnit – responsável pela simulação dos preços de um ativo.

5.4.1 PricingUnit

PrincingUnit é responsável pela precificação, recebe as cotações do ativo, dispara os argumentos da simulação e coleta os pacotes de simulações. O disparo dos argumentos e feito de modo a dividir a carga (cem mil simulações) entre as unidades de simulação. Sua interação com o servidor de mensagens pode ser vista na Figura 5.7.

5.4.2 SimulationUnit

Este componente é responsável pelas simulações do preço do ativo usando o modelo Black-Scholes e método Monte Carlo. A cada mensagen de argumento recebida retorna uma mensagem de simulações agregadas.

Cada nova mensagem de argumentos é independente, além dos parâmetros do modelo contém a quantidade de simulações a serem realizadas. Após realizadas as simulações, preprocessa estas amostras (somatório das amostras, $E(X)$, e somatório das amostras ao quadrado, $E(X^2)$) e publica este produto agregado na forma de uma mensagen. A Figura 5.8 esquematiza a interação da unidade de simulação com o servidor de mensagens.

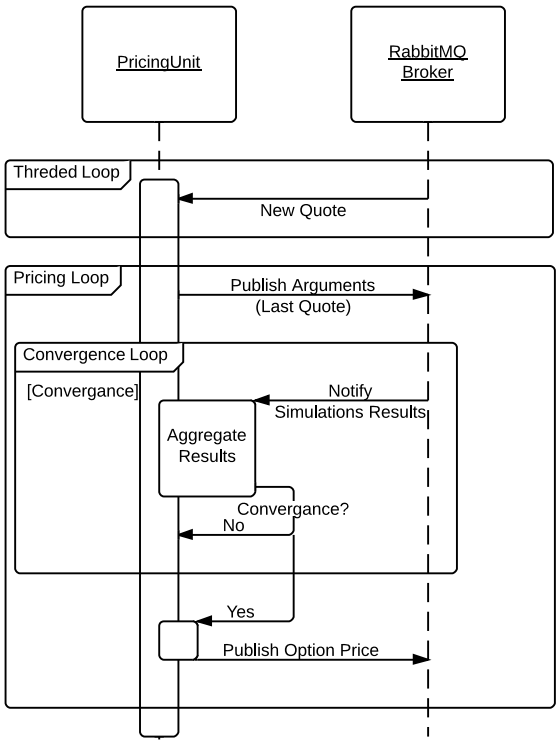

Figura 5.7: Diagrama de sequência de PricingUnit

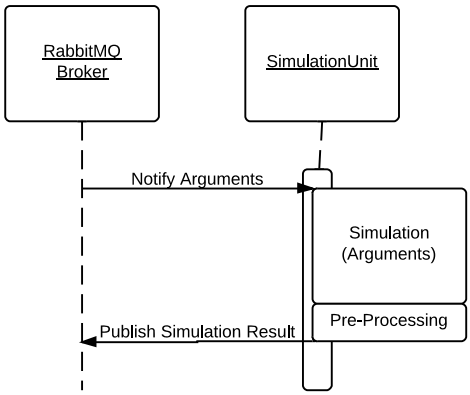

Figura 5.8: Diagrama de sequência de SimulationUnit

Na Figura 5.9 é descrita uma sequência hipotética de eventos da precificação de opções. Contando com a iteração de uma PricingUnit e cinco SimulationUnits com o servidor RabbitMQ. Note que o número de simulações em cada SimulationUnit ficou dividido por cinco $(100.000/5)$.

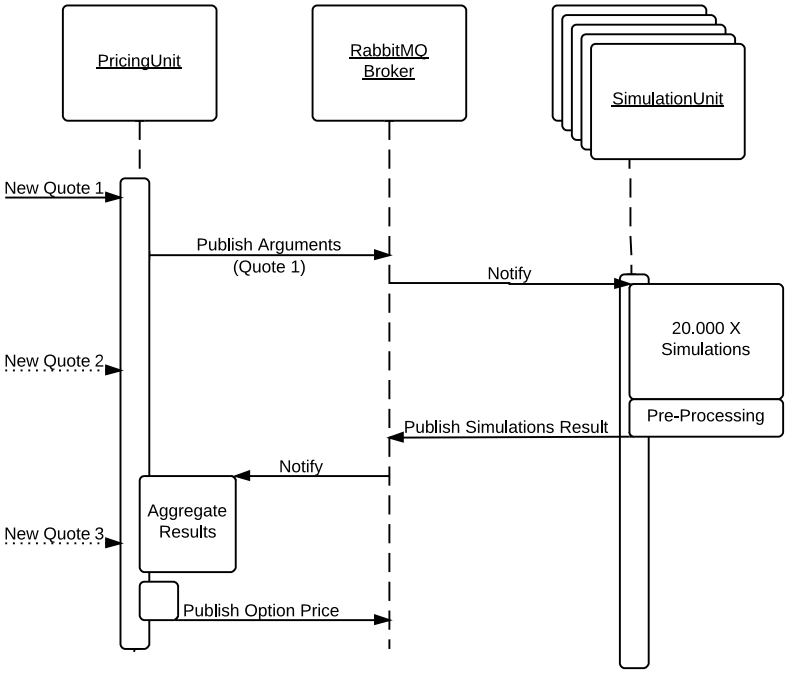

Figura 5.9: Sequência hipotética da precificação de opções

8

5.5 ProcessStarter

ProcessStarter é uma série de componentes implementados em Java para iniciar, gerenciar e monitorar componentes em máquinas remotas via mensagens AMQP, sistema esquematizado na Figura 5.10.

Componentes inicializados nas máquinas remotas (ProcessStarterClient) recebem e executam comandos de shell enviados pelo ProcessStarterManager. Usando software de sincronização de arquivos, como Dropbox, é possível manter uma pasta comum entre as máquinas remotas. Assim esta pasta pode ser usada para desenvolvimento, teste e repositório de releases de componentes distribuídos.

A sincronização de arquivos, a possibilidade de executar comandos shell em máquinas remotas e o serviço de mensagens AMQP, criam a sensação da extensão do sistema operacional. Idealmente um comando seria executado pela máquina com menor uso de processamento ou memória, mas neste protótipo as mensagens que contém os comandos são distribuídas de modo *round-robin*. ProcessStarter é constituído de três classes de componentes:

- ProcessStarterClient: Executado em máquinas remotas, recebe comandos e os executa.

- ProcessStarterManager: Software com interface gráfica para iniciar, gerenciar e monitorar processos remotos.

- ProcessStarterAPI: Biblioteca em Java, para iniciar processos remotos.

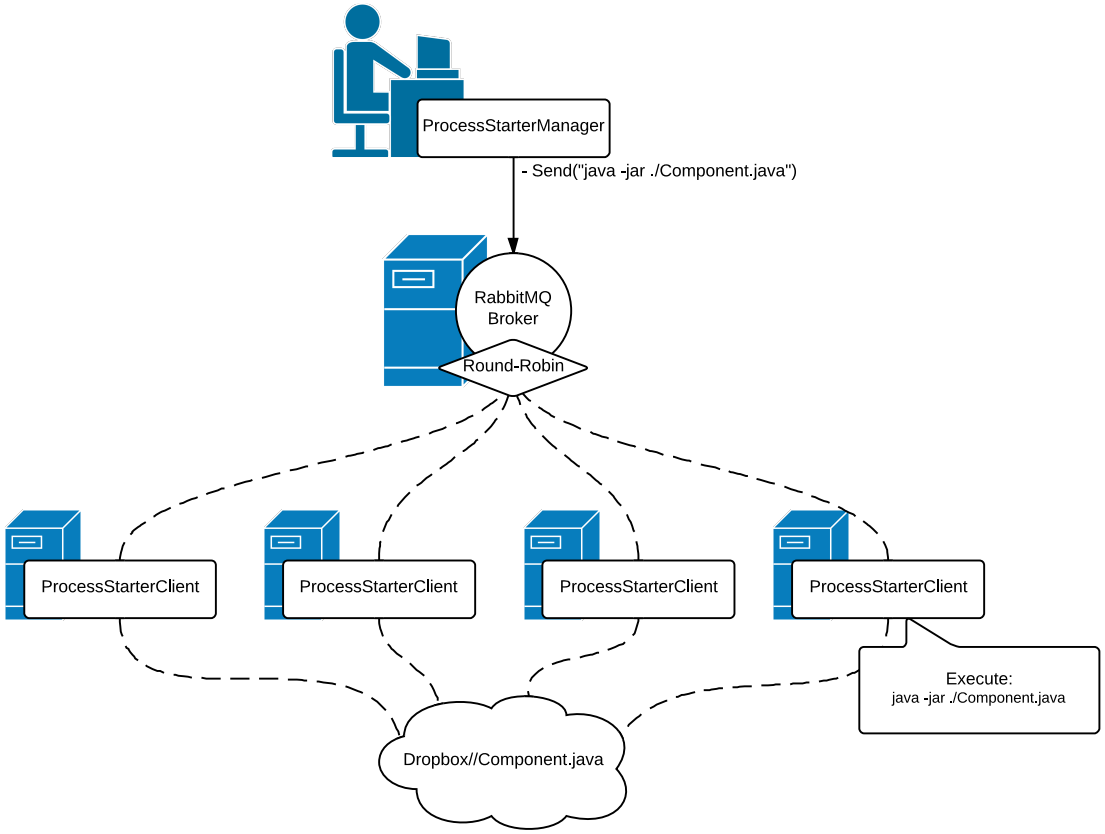

Figura 5.10: Diagrama do ProcessStarter

5.5.1 ProcessStarterClient

ProcessStarterClient é um componente iniciado em uma máquina remota, é identificado pelo IP:PID. Tem duas funcionalidades: Executar comandos shell enviados à "workin queu!hte" processStarter.command; Publicar o uso de processamento, uso memória da máquina e informações sobre processos(componentes) iniciados no exchange 'processstarter' com o tópico 'log'.

Usa uma software de sincronização de arquivos para manter uma pasta base de componentes comum a todas máquinas remotas. Os comandos são executados com raíz nesta pasta.

5.5.2 ProcessStarterManager

ProcessStarterManager é um aplicativo para enviar comandos aos ProcessStarterClient, e visualizar os dados publicados por estes. A captura de tela feita durante os experimentos, Figura 5.11, demonstra o ProcessStarterManager gerenciando dez ProcessStarterClient. No campo *Report* mostra as informações relevantes enviadas pelo cliente selecionado, como: processos iniciados, uso de CPU, memória, etc...

5.5.3 ProcessStarterAPI

É uma API implementada em Java que funciona similarmente a biblioteca java.lang.Runtime. Utiliza a interface ProcessStarter para iniciar processos remotos. Na Figura 5.12 é exemplificado o uso das bibliotecas java.lang.Runtime e ProcessStarterAPI.

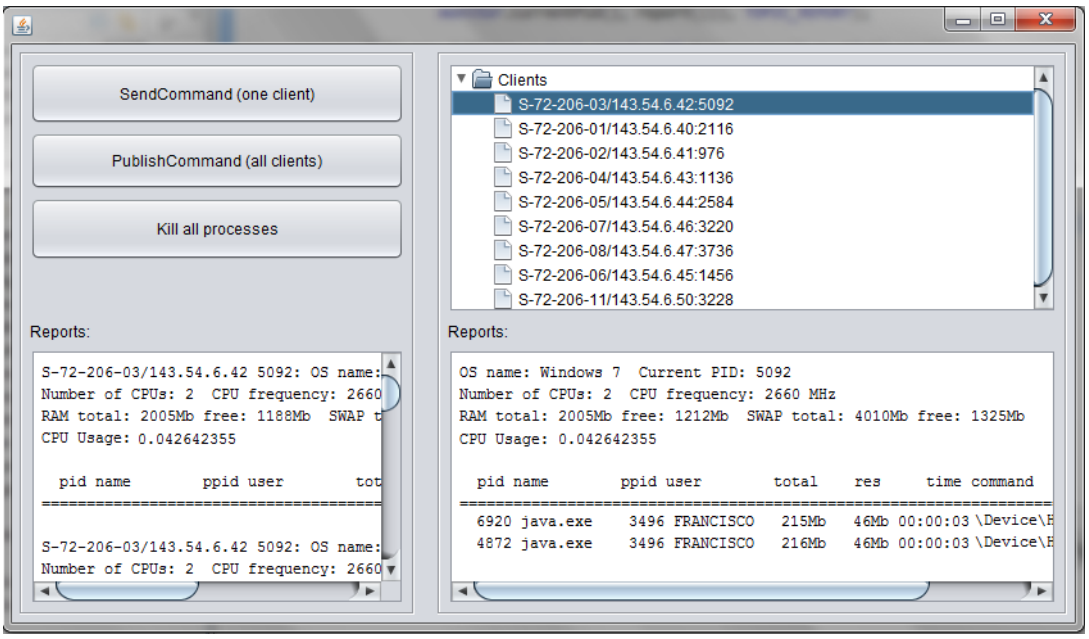

Figura 5.11: Captura de tela do ProcessStarterManager

```
1  // Execute calc.exe locally
2  Process p = Runtime.getRuntime().exec("calc");
3
4  // Execute calc.exe on a remote machine
5  RemoteProcess rp = ProcessStarter.execRemote("calc");
```

Figura 5.12: Exemplo de uso de java.lang.Runtime e ProcessStarterAPI

6 EXPERIMENTOS

Nesta seção serão apresentados testes de componentes e aplicações implementadas neste trabalho. Os resultados são a média de 30 repetições. No servidor de mensagens foi usado Windows 7 e nas máquinas clientes Linux (Ubuntu 12.10). Os objetivos dos experimentos se dividem em:

- XmlSerializer: testar o desempenho e tamanho da serialização.
- Componentes distribuídos: testar a desempenho e escalabilidade deste protótipo essencial ao framework.
- Avaliação de opções: testar a corretude, desempenho e escalabilidade deste produto financeiro implementado sobre o framework.

6.1 XmlSerializer

Na serialização existe dois importantes critérios de avaliação: velocidades de serialização e tamanho de objetos serializados. Para uma boa avaliação, será exposto os resultados dos serializadores XmlSerializer e Protocol Buffers[1](lançado em 2011 pela Google). A estrutura de dados MqlRates[2] (Figura 6.1), comum no mercado financeiro, foi usada para a avaliação.

```
1  message MqlRates {
2      required string symbol; // Asset symbol
3      required string time;   // Period start time
4      required double open;   // Open price
5      required double high;   // The highest price of the period
6      required double low;    // The lowest price of the period
7      required double close;  // Close price
8      required int64  tick_volume;  // Tick volume
9      required int32  spread;       // Spread
10     required int64  real_volume;  // Trade volume
11 }
```

Figura 6.1: Estrutura MqlRates

[1]http://code.google.com/p/protobuf/
[2]http://www.mql5.com/en/docs/constants/structures/mqlrates

Tabela 6.1: Tamanho do MqlRates serializado

Tipo de Serialização	Bytes
XmlSerializer	360
Protocol Buffers textual	236
Protocol Buffers binário	98

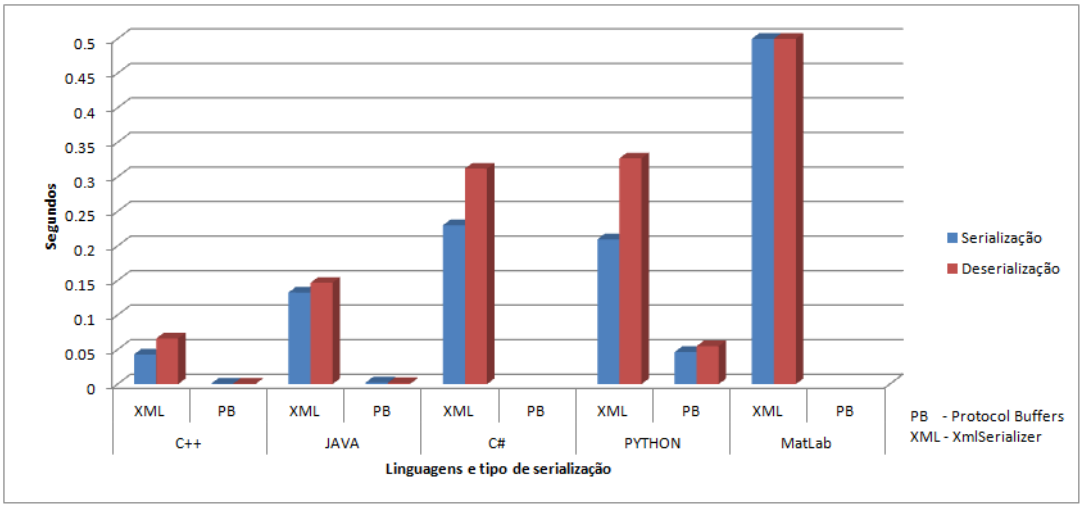

Figura 6.2: Tempo de serialização de 1000 objetos MqlRates

Os testes realizados de velocidade de serialização, Figura 6.2, foram feitos em um laptop Intel Core i3 2.13GHz, Windows 7. Protocol Buffers não fornece suporte ao MatLab ou C#. Os resultados da serialização XML no MatLab alcançaram valores altos (aproximadamente 4 segundos), logo foram truncados para permitir melhor visualização do gráfico.

XmlSerializer gera serialização de maior tamanho e com menor velocidade que Protocol Buffers. Suas vantagens estão na maior abrangência de interoperabilidade e simplicidade de uso. Em C++ a classe MqlRates em serialização XML apresenta 10 linhas, ProtocolBuffer mais de 1000 linhas (a mesmo acontece em Java). O sistema XmlSerializer pode ser aplicado com poucas linhas de código adicionais em classes já definidas, Protocol Buffers necessita a criação de classes especiais para a comunicação. XmlSerializer é um sistema de serialização intuitivo pois seu uso não implica mudanças significativas no arquitetura do sistema ou método de programação.

6.2 Componentes distribuídos

Usando as classes e interfaces implementadas para o *framework*, em Java e Python, foi testada a taxa de consumo de mensagens de componentes distribuídos. Os testes seguintes foram feitos usando um laptop Intel Core i3 2.13GHz, Windows 7, como servidor de mensagens RabbitMQ e máquinas do laboratório de informática do campus do vale da UFRGS como cluster, Intel Core 2 Duo E6750 2.6GHz. Foi usada a rede *wireless* para a comunicação do servidor de mensagens e máquinas do laboratório. Este fato afetou negativamente os testes latência de mensagens no *framework*. Os experimentos de avaliação de opções ficaram menos afetados pois possuem alta carga de processameto e poucas trocas de mensagens.

Primeiramente foi testada a latência *round-trip* de 1000 mensagens MqlRates em *localhost*, instanciando uma unidade produtora e uma processadora na mesma máquina do servidor de mensagens. A unidade processadora realiza uma operação em memória antes de enviar a mensagem de volta a produtora. As unidades usam XmlSerilizer e RabbitMQ em suas interfaces.

Tabela 6.2: *Round-trip* de 1000 mensagens em *localhost*

Linguagem	Tempo Total	Taxa de processamento
Java	0,492 seg	2032 msg/seg
Python	1,758 seg	568 msg/seg

No experimento da Figura 6.3 testamos o *round-trip* de mensagens com diversos consumidores, um para cada máquina remota. Se ressalta novamente o efeito negativo da baixa qualidade na comunicação do ambiente de teste, a razão de tempo no teste *round-trip* entre um consumidor remoto e um consumidor em *localhost* é de $8,8$.

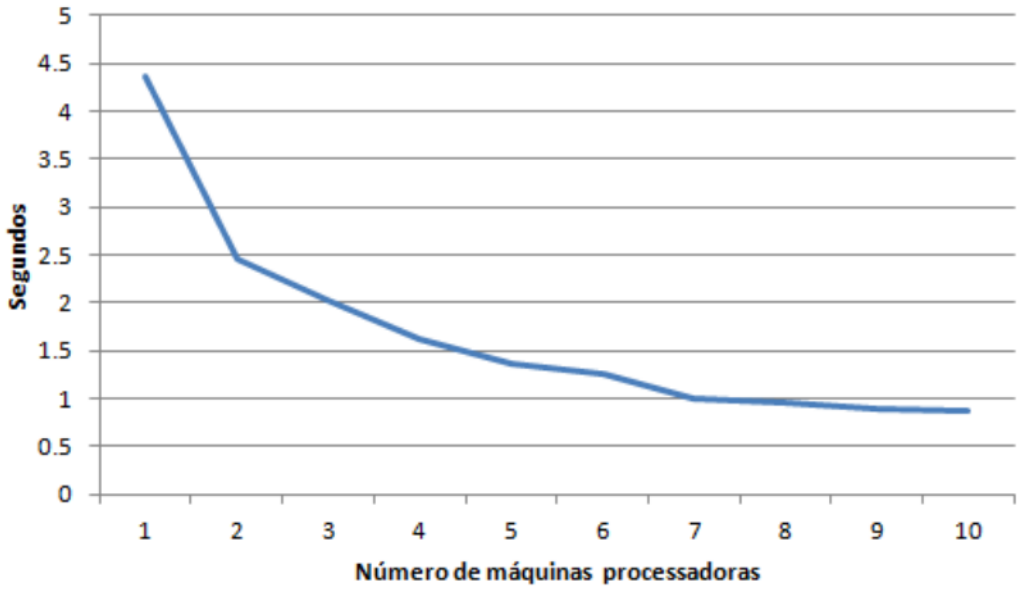

Figura 6.3: Round-trip de 1000 mensagens em ambiente distribuído

Os resultados mostram ganhos expressivos no decorrer do aumento de máquinas, mas estes ganhos diminuem devido a uma sobrecarga na rede. Após 8 máquinas simultâneas a otimização é quase imperceptível, no entanto, é possível dividir o trabalho de processamento no lado do servidor de mensagens, por clusterização do servidor, e no recebimento final das mensagens aumentando o desempenho da escalabilidade e do tempo de execução.

6.3 Avaliação de opções

6.3.1 Corretude

Primeiro é verificado se os resultados estão corretos usando os mesmos parâmetros de (Chorro 2008), S = 100, E = 100, σ = 0,15, r = 0,05, t = 0, δt = 1. Na Tabela 6.3 observamos a corretude do experimento pelas amostras obtidas na avaliação (são amostras

escolhidas arbitrariamente, o modelo Black-Scholes não é preciso suficiente para convergir sempre ao mesmo valor).

Tabela 6.3: Valores estimados e intervalos de confiança da avaliação de opções

(Chorro 2008)	
Valor Estimado	IC (95%)
8,68	[8,49; 8,87]
8,88	[8,82; 8,95]

Valores obtidos	
Valor Estimado	IC (95%)
8,80	[8,73; 8,87]
8,63	[8,56; 8,70]

6.3.2 Paralelização

Para avaliar o desempenho de paralelização da avaliação de opções sobre o framework realizamos dois experimentos:

- Alto desempenho: distribuição da avaliação de uma opção em diferentes quantidades de máquinas.
- Escalabilidade: distribuição da avaliação de diferentes quantidades de opções concorrentemente em dez máquinas.

Inicialmente foi simulado o método Monte Carlo com 200 e 400 passos no preço de modo sequencial em uma das máquinas clientes, os resultados obtidos foram:

Tabela 6.4: Tempo da execução do método Monte Carlo não paralelizado

Passos	Tempo Total
200	3,56 seg
400	7,16 seg

Cada máquina do "cluster" tem dois núcleos (Intel Core 2 Duo), a fim de utilizar ao máximo a sua capacidade de processamento, no modelo distribuído instanciamos duas unidades SimulationUnit por máquina. No gráfico da Figura 6.4 podemos notar, em ambos os casos, que a avalição distribuída por mais de quatro máquinas (oito SimulationUnit) não alcança ganhos expressivos. O caso 0,5 é uma máquina remota com apenas uma SimulationUnit.

Sendo o *speedup* definido como $Sp = T1/Tp$ (tempo de execução sequencial sobre paralelizado) e a eficiência $Ep = Sp/p$ (*speedup* sobre numero de processadores). Com 9 máquinas temos *speedup* de 8,3 e eficiência de 46%. Com 4 máquinas temos *speedup* de 5,9 e eficiência de 73%. Os *speedups* obtidos são proporcionais aos constatados por (Tian e Benkrid 2010) na mesma proporção de processadores em um servidor de múltiplos núcleos Intel Xenon, Tabela 6.5.

Tabela 6.5: Comparação de *speedups* na avaliação de opções

Experimento	2 CPUs	4 CPUs	8 CPUs	16 CPUs
Framework	1,87	3,47	6,8	8,3
(Tian e Benkrid 2010)	1,95	3,9	7,9	11,2

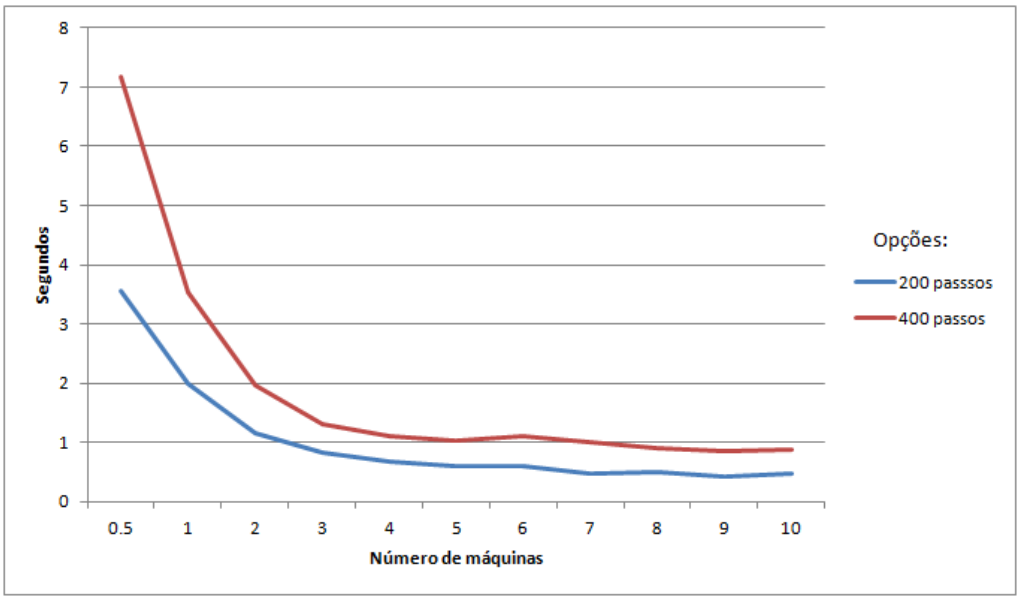

Figura 6.4: Distribuição da avaliação de uma opção em diferentes quantidades de máquinas

O teste para avaliar o desempenho de opções concorrentes foi criado de maneira que todas as unidades PricingUnit iniciassem suas avaliações de maneira simultânea. Depois de iniciada cada unidade repete o processo de avaliação 30 vezes e retorna a média do tempo de avaliação. O gráfico da Figura 6.5 mostra o tempo de execução e a relação entre o número de opções concorrentes e o numero divisor das simulações entre as máquinas do cluster.

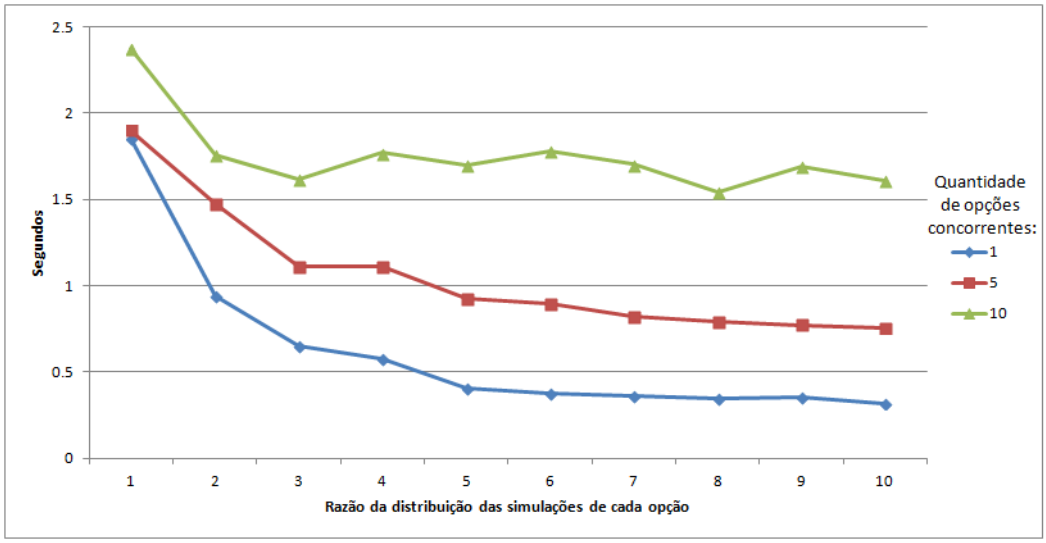

Figura 6.5: Avaliação de opções concorrentes em 10 máquinas

Nota-se que foram alcançados tempos de avaliação menores para 10 avaliações concorrentes em 10 máquinas do que 1 avaliação em 1 máquina. Estes valores são apresentados na Tabela 6.6, são uma evidência forte da escalabilidade do framework para produtos financeiros de alto desempenho.

Tabela 6.6: Valores estimados e intervalos de confiança da avaliação de opções

Razão de uma máquina por opção	
Opções	Tempo estimado (seg)
1	1,85
5	1,9
10	1,54

Razão de duas máquinas por opção	
Opções	Tempo estimado (seg)
1	0,94
5	0,75

7 CONCLUSÃO

Produtos financeiros complexos necessitam de uma alternativa barata e confiável para mapear seu processamento em arquiteturas *many-core* de alto desempenho, como *grids* e *clouds*. Neste trabalho foi apresentado um framework para facilitar a criação de sistemas distribuídos na área financeira. O modelo usa padrões como Component-Based Development e Service Oriented Architecture para prover uma maior integração de software. Utilizando tecnologias abertas como Advanced Message Queuing Protocol e serialização XML foram criadas classes e interfaces que auxiliam no projeto, desenvolvimento e implantação de componentes interoperáveis de baixa latência.

O framework demonstrou bom desempenho e alto poder de escalabilidade nos experimentos realizados. A serialização XML é uma alternativa altamente interoperável e a comunicação assíncrona por servidor de mensagens minimiza a redundância de dados e acoplamento entre comunicantes na rede. Nos experimentos de carga de mensagens e avaliação de opções foi notada a sobrecarga da rede em um determinado limite, após o qual, o aumento de clientes simultâneos não melhora a velocidade. No entanto, é possível dividir o trabalho de processamento no lado do servidor de mensagens, por clusterização do servidor, e no recebimento final das mensagens aumentando o desempenho da escalabilidade e do tempo de execução.

Em um ambiente distribuído e escalável o framework conseguiu, na mesma aplicação, *speedup* similar a um Intel Xenon com a mesma proporção de processadores. Os experimentos poderiam ser aprimorados utilizando uma rede livre de interferências externas e de maior banda. Um sistema com apenas um servidor de mensagens cria um ponto único de falha. Em trabalhos futuros seria interessante testar o comportamento de um servidor clusterizado ou em federação e analisar seu potencial de amenização da sobrecarga.

Frameworks similhares que usam comunicação AMQP, interfaces por serialização e componentes distribuídos podem ser encontrados nos trabalhos (Keetha e He 2009) e (Arkhipkin e Lauret 2011).

REFERÊNCIAS

[Arkhipkin e Lauret 2011]ARKHIPKIN, D.; LAURET, J. A message-queuing framework for star's online monitoring and metadata collection. *Journal of Physics: Conference Series*, v. 331, n. 2, p. 022003, 2011. Available from Internet: <http://stacks.iop.org/1742-6596/331/i=2/a=022003>.

[Armstrong et al. 1999]ARMSTRONG, R. et al. Toward a common component architecture for high-performance scientific computing. In: *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 1999. (HPDC '99), p. 13–. ISBN 0-7695-0287-3. Available from Internet: <http://dl.acm.org/citation.cfm?id=822084.823232>.

[BM&FBOVESPA 2012]BM&FBOVESPA. *O Mercado de Opções*. 2012. Available from Internet: <http://www.bmfbovespa.com.br/>.

[Box e Muller 1958]BOX, G.; MULLER. A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, v. 29, p. 610–611, 1958.

[Boyle 1977]BOYLE, P. P. Options: A monte carlo approach. *Journal of Financial Economics 4*, 1977.

[Caruana 2010]CARUANA, J. Why basel iii matters for latin american and caribbean financial markets. *Bank of International Settlements*, 2010.

[Chorro 2008]CHORRO, C. *Monte Carlo Methods and Black Scholes model*. 2008. Available from Internet: <http://christophe.chorro.fr/docs/Malliavin1.pdf>.

[Cisco 2008]CISCO, S. I. *Trading Floor Architecture*. 2008.

[Crnkovic e Chaudron 2006]CRNKOVIC, I.; CHAUDRON, M. Component-based development process and component lifecycle. In: *Proceedings of the International Conference on Software Engineering Advances*. Washington, DC, USA: IEEE Computer Society, 2006. (ICSEA '06), p. 44–. ISBN 0-7695-2703-5. Available from Internet: <http://dx.doi.org/10.1109/ICSEA.2006.28>.

[Curry 2004]CURRY, E. *Message-Oriented Middleware*. [S.l.]: John Wiley and Sons, 2004.

[Ellis 1997]ELLIS, D. M. Different sides of the same story: Investors? and issuers? views of rating agencies. *SSRN*, 1997. Available from Internet: <http://ssrn.com/abstract=40680>.

[Ennis e Price 2011]ENNIS, H. M.; PRICE, D. A. Basel iii and the continuing evolution of bank capital regulation. *The Federal Reserve Bank of Richmond*, 2011.

[GigaSpaces 2004]GIGASPACES, T. L. *GigaSpaces – Merrill Lynch Case Study*. 2004.

[Hohpe e Woolf 2003]HOHPE, G.; WOOLF, B. *Enterprise Integration Patterns, Designing, Building and Deploying Messaging Solutions.* [S.l.]: Addison-Wesley Professional, 2003.

[Homer e Sylla 1996]HOMER, S.; SYLLA, R. *A history of interest rates*. [S.l.]: Wiley Finance, 1996.

[IBM 2006]IBM, I. f. B. V. *IBM Financial Markets Bussines Survey*. 2006.

[Irturk et al. 2008]IRTURK, A. et al. Fpga acceleration of mean variance framework for optimal asset allocation. In: *High Performance Computational Finance, 2008. WHPCF 2008. Workshop on*. [S.l.: s.n.], 2008. p. 1 –8.

[Johnson 2008]JOHNSON, P. Improved numerical techniques for occupation-time derivatives and other complex financial instruments. *Manchester Institute for Mathematical Sciences, The University of Manchester*, 2008. Available from Internet: <http://eprints.ma.man.ac.uk/1357/>.

[Kaelber 2004]KAELBER, L. Max weber on usury and medieval capitalism: From the history of commercial partnerships to the protestant ethic. *EBSCOhost*, 2004.

[Kajko-Mattsson e Lewis 2007]KAJKO-MATTSSON, M.; LEWIS, G. A framework for roles for development, evolution and maintenance of soa-based systems. In: *Proceedings of the International Workshop on Systems Development in SOA Environments*. Washington, DC, USA: IEEE Computer Society, 2007. (SDSOA '07), p. 7–. ISBN 0-7695-2960-7. Available from Internet: <http://dx.doi.org/10.1109/SDSOA.2007.1>.

[Keetha e He 2009]KEETHA, N. R.; HE, K. Hiperfs: A framework for high performance financial services using advanced message queuing. 2009. Available from Internet: <www1.cs.columbia.edu>.

[Klein 2011]KLEIN, H. Amqp in financial services - deutsche börse group. In: *Fourth Annual AMQP Conference*. [S.l.: s.n.], 2011.

[Krogdahl e Luef 2005]KROGDAHL, P.; LUEF, G. Service-oriented agility: an initial analysis for the use of agile methods for soa development. In: *Services Computing, 2005 IEEE International Conference on*. [S.l.: s.n.], 2005. v. 2, p. 93 – 100 vol.2.

[Levina e Stantchev 2009]LEVINA, O.; STANTCHEV, V. Realizing event-driven soa. In: *Internet and Web Applications and Services, 2009. ICIW '09. Fourth International Conference on*. [S.l.: s.n.], 2009. p. 37 –42.

[Liukkonen 2009]LIUKKONEN, T. *Human Reaction Times as a Response to Delays in Control Systems*. 2009.

[Maréchaux 2006]MARéCHAUX, J.-L. *The Common Object Request Broker: Architecture and Specification, Revision 1.1*. 2006. Available from Internet: <http://www.ibm.com/developerworks/library/ws-soa-eda-esb/>.

[Microsoft 2012]MICROSOFT, m. *Serializing Objects*. 2012. Available from Internet: <http://msdn.microsoft.com/en-us/library/7ay27kt9(v=vs.71).aspx>.

[MPI 1995]MPI, F. *MPI: A Message-Passing Interface Standard*. Message Passing Interface Forum, 1995. Available from Internet: <http://www.mpi-forum.org>.

[Niramarnsakul e Chongstitvatana 2011]NIRAMARNSAKUL, C.; CHONGSTITVATANA, P. Parallelization of european monte-carlo options pricing on graphics processing units. In: *Computer Science and Software Engineering (JCSSE), 2011 Eighth International Joint Conference on*. [S.l.: s.n.], 2011. p. 247 –249.

[O'Hara 2007]O'HARA, J. Toward a commodity enterprise middleware. *Queue*, ACM, New York, NY, USA, v. 5, n. 4, p. 48–55, maio 2007. ISSN 1542-7730. Available from Internet: <http://doi.acm.org/10.1145/1255421.1255424>.

[OMG 1995]OMG, O. M. G. *The Common Object Request Broker: Architecture and Specification, Revision 1.1*. 1995. Available from Internet: <http://www.omg.org/>.

[RabbitMQ 2012]RABBITMQ, V. *RabbitMQ interfaces*. 2012.

[Saariluoma 1995]SAARILUOMA, P. *Chess players' thinking: a cognitive psychological approach*. 1995.

[Schryver et al. 2011]SCHRYVER, C. d. et al. An energy efficient fpga accelerator for monte carlo option pricing with the heston model. In: *Proceedings of the 2011 International Conference on Reconfigurable Computing and FPGAs*. Washington, DC, USA: IEEE Computer Society, 2011. (RECONFIG '11), p. 468–474. ISBN 978-0-7695-4551-6. Available from Internet: <http://dx.doi.org/10.1109/ReConFig.2011.11>.

[Senate 2011]SENATE, U. S. *WALL STREET AND THE FINANCIAL CRISIS: Anatomy of a Financial Collapse*. 2011. Available from Internet: <www.hsgac.senate.gov>.

[Sessions 1997]SESSIONS, R. *COM and DCOM: Microsoft's Vision for Distributed Objects*. [S.l.]: John Wiley and Sons, 1997.

[Smelyanskiy 2008]SMELYANSKIY, M. Challenges of mapping financial analytics to many-core architecture. In: *High Performance Computational Finance, 2008. WHPCF 2008. Workshop on*. [S.l.: s.n.], 2008. p. 1.

[Smith 1927]SMITH, J. G. *A Forgotten Chapter in the Early History of the Corporate Trust Deed*. [S.l.]: American Law Review, 1927.

[Tanenbaum e Steen 2002]TANENBAUM, A. S.; STEEN, M. V. *Distributed Systems: Principles and Paradigms*. [S.l.: s.n.], 2002.

[Tian e Benkrid 2010]TIAN, X.; BENKRID, K. High-performance quasi-monte carlo financial simulation: Fpga vs. gpp vs. gpu. *ACM Trans. Reconfigurable Technol. Syst.*, ACM, New York, NY, USA, v. 3, n. 4, p. 26:1–26:22, nov. 2010. ISSN 1936-7406. Available from Internet: <http://doi.acm.org/10.1145/1862648.1862656>.

[Vinoski 2006]VINOSKI, S. Advanced message queuing protocol. *IEEE Internet Computing*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 10, n. 6, p. 87–89, nov 2006. ISSN 1089-7801. Available from Internet: <http://dx.doi.org/10.1109/MIC.2006.116>.

[Wieland et al. 2009]WIELAND, M. et al. Soeda: A method for specification and implementation of applications on a service-oriented event-driven architecture. In: ABRAMOWICZ, W. (Ed.). *Business Information Systems*. Springer Berlin Heidelberg, 2009, (Lecture Notes in Business Information Processing, v. 21). p. 193–204. ISBN 978-3-642-01189-4. Available from Internet: <http://dx.doi.org/10.1007/978-3-642-01190-0_17>.

[Ying 2011]YING, X. A study on risk management technology of commercial banks. In: *E -Business and E -Government (ICEE), 2011 International Conference on*. [S.l.: s.n.], 2011. p. 1 –3.